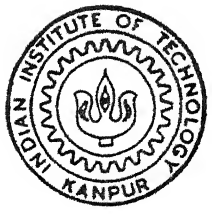
MULTIOBJECTIVE OPTIMIZATION USING NONDOMINATED SORTING IN GENETIC ALGORITHMS

by

Nidamarthi Srinivas



DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JANUARY, 1994

MULTIOBJECTIVE OPTIMIZATION USING NONDOMINATED SORTING IN GENETIC ALGORITHMS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

Nidamarthi Srinivas

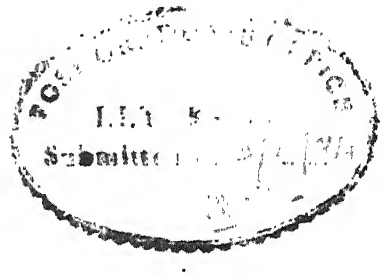
to the

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

January, 1994

ME-1994-M-SRI-MUL

Dedicated
to my
parents



CERTIFICATE

It is certified that the work contained in the thesis entitled "**Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms**", by *Nidamarthi Srinivas*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

(Kalyanmoy Deb)
Assistant Professor
Department of Mech. Engg.
I. I. T. Kanpur

(Sanjay G. Dhande)
Professor & Head
Department of Mech. Engg.
I. I. T. Kanpur

January, 1994

ACKNOWLEDGEMENTS

I am indebted to Dr. Kalyanmoy Deb, for introducing me to the field of genetic algorithms and providing guidance throughout my graduation. A special thanks is due Dr. S. G. Dhande, whose guidance and expertise have been considerable help to me. I would also like to thank Dr. Amitabh Mukerjee and Dr. Brahma Deo, for their valuable comments and suggestions about this work.

I am greatful to my parents and sisters, whose love and encouragement helped me overcome many difficulties during my graduation. Finally I would like to thank all my friends, especially Chandra Sekhar and Sarma, who have made my stay here an immensely educative and memorable experience.

Nidamarthi Srinivas

ABSTRACT

In a typical multiobjective optimization problem, solution exists in the form of a set of points, known as nondominated or Pareto-optimal points. In trying to solve multiobjective optimization problems, most traditional methods scalarize the objective vector into a single objective. In those cases, the obtained solution is highly sensitive to the weight vector used in the scalarization process and demands the user to have knowledge about the underlying problem. Moreover, in solving multiobjective problems, designers may be interested in a set of Pareto-optimal points, instead of a single point. Since genetic algorithms(GAs) work with a population of points, it seems natural to use GAs in multiobjective optimization problems to capture a number of solutions simultaneously. Although a vector evaluated GA (VEGA) has been implemented by J. D. Schaffer and has been tried to solve a number of multiobjective problems, the algorithm seems to have potential bias against middling individuals. As a result, its population tended to drift towards individual optima of objectives. A new algorithm, Nondominated Sorting Genetic Algorithm (NSGA), is developed and implemented in this thesis based on David Goldberg's suggestion. This algorithm not only takes care of the unavoidable bias in VEGA but also distributes the population over Pareto-optimal regions. This helps to find multiple Pareto-optimal points in a single run. The proof-of-principle results obtained on three test problems used by Schaffer and others, show a promising future of NSGA. NSGA is also applied to two real-world engineering problems. A number of suggestions for extension and application of the algorithm is also discussed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most real-world design or decision making problems involve simultaneous optimization of multiple objectives. In principle, multiobjective optimization is very different than the single-objective optimization. In single objective optimization, one attempts to obtain the *best* design or decision, which is usually the global minimum or the global maximum depending on the optimization problem is that of minimization or maximization. In the case of multiple objectives, there may not exist one solution which is best (global minimum or maximum) with respect to all objectives. In a typical multiobjective optimization problem, there exists a set of solutions which are superior to the rest of solutions in the search space in all objectives but are inferior to any other solution in the set in one or more objectives. These solutions are known as *Pareto-optimal* solutions or *nondominated* solutions (Chankong and Haimes, 1983; Hans, 1988). The rest of the solutions are known as dominated solutions. Since none of solutions in the nondominated set is *absolutely* better than any other, any one of them is theoretically an optimal solution. The choice of one solution over the other requires problem knowledge and a number of problem-related factors. Thus, one solution chosen by a designer may not be acceptable to another designer or in a changed environment. Therefore, in multiobjective optimization problems, it may be useful to have a knowledge about alternative Pareto-optimal solutions.

In trying to solve multiobjective problems, most traditional methods scalarize a vector of objectives into one objective by averaging the objectives with a weight vector. This process allows a simpler optimization algorithm to be used, but the obtained solution highly depends on the weight vector used in the scalarization process. Moreover, if available, a decision maker may be interested in knowing alternate solutions. Since genetic algorithms (GAs) work on population of points, a number of Pareto-optimal solutions may be captured using GAs. An early GA application on multiobjective optimization by J. D. Schaffer (1984) opened a new avenue of research in this field. Though his algorithm, VEGA, gave encouraging results, it suffered from its biasness against some Pareto-optimal solutions.

A new algorithm, Nondominated Sorting Genetic Algorithm (NSGA), is developed and implemented in this thesis based on David Goldberg's suggestion (Goldberg, 1989). This algorithm not only takes care of the unavoidable bias in VEGA but also distributes the population over Pareto-optimal regions. This thesis briefly describes the difficulties of using three common classical methods to solve multiobjective optimization problems. A brief introduction to Schaffer's VEGA and its problems are outlined. Thereafter, the nondominated sorting GA is described and applied to three test problems and two real world engineering problems. Experimental results for these test functions are compared and analyzed with that of VEGA. Chapter 2 introduces multiobjective optimization and concepts of Pareto-optimality. It also presents three classical methods and their drawbacks. Chapter 3 introduces genetic algorithms in function optimization. Chapter 4 discusses Schaffer's vector evaluated genetic algorithm and its shortcomings. Chapter 5 presents the nondominated sorting genetic algorithm, NSGA, and its implementation. Simulation results of three test functions and performance analysis in case of VEGA and NSGA arer presented in Chapter 6. Chapter 7 presents the successful application of NSGA on two engineering multiobjective optimization problems. Chapter 8 discusses the possible extensions of this algorithm. Conclusions are presented in chapter 9.

Chapter 2

Multiobjective Optimization - Classical Methods

This chapter introduces the multiobjective optimization problem and discusses three classical methods to solve multiobjective optimization problems. It also presents the difficulties of using these methods.

2.1 Multiobjective Optimization Problem

A general multiobjective optimization problem consists of a number of objectives and is associated with a number of inequality and equality constraints. Mathematically, the problem can be written as follows (Rao, 1991):

$$\begin{array}{rll} \text{Minimize/Maximize} & f_i(\mathbf{x}) & i = 1, 2, \ldots, N \\ \text{Subject to} & g_j(\mathbf{x}) \leq 0 & j = 1, 2, \ldots, J \\ & h_k(\mathbf{x}) = 0 & k = 1, 2, \ldots, K \end{array} \tag{2.1}$$

The parameter $\mathbf{x}$ is a p dimensional vector having p design or decision variables. In a single-objective optimization problem, one can find a unique point which is a global optimum. But in this vector optimization problem, solution exists in the form of a set of *satisfying* solutions. This is because there is always an alternative which is better in one objective or other. This concept is further discussed in the following section.

2.2 Concept of Nondominance

Solutions to a multiobjective optimization problem are mathematically expressed in terms of nondominated or superior points. In a minimization problem, a vector $\mathbf{x}$ is partially less than another vector $\mathbf{y}$, $(\mathbf{x} \prec \mathbf{y})$, when no value of $\mathbf{y}$ is less than $\mathbf{x}$ and at least one value of $\mathbf{y}$ is strictly greater than $\mathbf{x}$. In other words,

$$(\mathbf{x}) \prec (\mathbf{y}) \iff (\forall_i)\ (x_i \leq y_i) \quad \text{and} \quad (\exists_i)\ (x_i < y_i).$$

If $\mathbf{x}$ is partially less than $\mathbf{y}$, we say that $\mathbf{x}$ *dominates* $\mathbf{y}$ or $\mathbf{y}$ is *inferior* to $\mathbf{x}$ (Tamura and Miura, 1979). Any member of such vectors which is not dominated by any other member is said to be *nondominated* or *non-inferior*. Computationally, we say $\mathbf{y}$ is dominated if the following conditions are satisfied :

$$\begin{aligned} f_i(\mathbf{x}) &\leq f_i(\mathbf{y}) \quad \text{for all } i, \text{ and} \\ f_i(\mathbf{x}) &< f_i(\mathbf{y}) \quad \text{for at least one } i, \end{aligned} \tag{2.2}$$

where f_i is i-th objective. It is important to note that the above conditions are true for minimization only. If the objective is to maximize a function we define a dominated point if the corresponding component is not greater than that of a nondominated point. In this case one has to redefine the equation 2.2 by simply changing the relational operator from less-than or less-than-equal-to type to greater-than or greater-than-equal-to type. This way, we can also extend this methodology to handle minimization of a set of functions at the same time maximizing another set. The optimal solutions to a multiobjective optimization problem are nondominated solutions. They are also known as *Pareto-optimal* solutions. Mathematically, an optimization algorithm should be terminated if any one of the Pareto-optimal solution is obtained. But in practice, since there could be a number of Pareto-optimal solutions and the suitability of one solution depends on a number of factors including designer's choice, problem environment, a knowledge of multiple Pareto-optimal solutions is desired.

The following section describes three classical approaches to the solution of multi-objective optimization problems and discusses their difficulties by illustrating a simple two-objective optimization problem.

2.3 Classical Methods

Even though there exists a number of techniques to solve multiobjective optimization problems, a common difficulty with multiobjective optimization is the appearance of an *objective conflict* (Hans, 1988)—none of the feasible solutions allow simultaneous optimal solutions for all objectives. In other words, individual optimal solutions of each objective are usually different. Thus, a mathematically most favorable Pareto-optimum is that solution which offers least objective conflict. But such solutions may not satisfy a decision-maker because he may want a satisfying solution according to the associated priorities of objectives. To find such points, all classical methods scalarize the objective vector into one objective. Most of the classical algorithms for non-linear vector optimization define a substitute problem, reducing the vector optimization to a scalar optimization. Using such substitute, a compromise solution is found subjected to specified constraints.

In the following subsections, three commonly-used methods—method of objective weighting, method of distance functions, and method of min-max formulation—are discussed.

2.3.1 Method of Objective Weighting

This is probably the simplest of all classical techniques. Multiple objective functions are combined into one overall objective function, Z, as follows:

$$Z = \sum_{i=1}^{\mathbf{N}} w_i f_i(\mathbf{x}), \tag{2.3}$$

where $\mathbf{x} \in \mathbf{X}$, the weights w_i are fractional numbers ($0 \leq w_i \leq 1$), and all weights are summed up to one, or $\sum_{i=1}^{\mathbf{N}} w_i = 1$. In this method, the optimal solution is controlled

by the weight vector $\mathbf{w}$. It is clear from above equation that the preference of an objective can be changed by modifying the corresponding weight. Mathematically, a solution obtained with equal weights to all objectives may offer least objective conflict, but as a real-world situation demands a satisfying solution, priority must be induced in the formulation. In most cases, each objective is first optimized and then the solutions obtained are used to choose the weight vector. The only advantage of using this technique is that the obtained solution is a guaranteed Pareto-optimum solution.

2.3.2 Method of Distance Functions

In this method, the scalarization is achieved by using a demand-level vector $\bar{\mathbf{y}}$ which has to be specified by the decision maker. The single objective function derived from multiple objectives is as follows:

$$Z = \left[\sum_{i=1}^{\mathbf{N}} |f_i(\mathbf{x}) - \bar{y}_i|^r\right]^{1/r}, \qquad 1 \leq r < \infty, \quad \mathbf{x} \in \mathbf{X}. \tag{2.4}$$

Usually an Euclidean metric $r = 2$ is chosen, with $\bar{\mathbf{y}}$ as individual optima of objectives (Hans, 1988). It is important to note that the solution obtained by solving above equation depends on the chosen demand-level vector. Arbitrary selection of a demand level may be highly undesirable. This is because a wrong demand level will lead to a non Pareto-optimal solution. As the solution is not guaranteed, the decision maker must have a thorough knowledge of individual optimum of each objective prior to the selection of demand level. In a way this method works as a goal programming technique imposing a goal vector, $\bar{\mathbf{y}}$ (demand level), on the given objectives. This method is similar to method of objective weighting with only difference in that in this method the goal for each objective function is required to be known and in the previous method the relative importance of each objective is required to be known.

2.3.3 Min-Max Formulation

This method is different in principle than the above two methods. This method attempts to minimize the relative deviations of the single objective functions from individual optimum. That is, it tries to minimize the objective conflict. For a minimization problem, the corresponding min-max problem is formulated as follows:

$$\text{minimize } \mathcal{F}(\mathbf{x}) = \max\,[Z_j(\mathbf{x})], \qquad j = 1, 2, \ldots, \mathbf{N}, \tag{2.5}$$

where $\mathbf{x} \in \mathbf{X}$ and $Z_j(\mathbf{x})$ is calculated for nonnegative target optimal value $\bar{f}_j > 0$ as follows:

$$Z_j(\mathbf{x}) = \frac{f_j - \bar{f}_j}{\bar{f}_j}, \quad j = 1, 2, \ldots, \mathbf{N}. \tag{2.6}$$

This method can yield best possible compromise solution when objectives with equal priority are required to be optimized. However, priority of each objective can be varied by introducing dimension-less weights in the formulation. This can also be modified as a goal programming technique by introducing a demand-level vector in the formulation.

2.4 Drawbacks of Classical Methods

In all above methods, multiple objectives are combined to form one objective by using some knowledge of the problem being solved. The optimization of the single objective may guarantee a Pareto-optimal solution but results in single point solution. In real world situations decision makers need different alternatives in decision making. Moreover, if some of the objectives are noisy or have discontinuous variable space these methods may not work effectively. Some of these methods are also expensive as they require individual optimum prior to vector optimization. The most profound drawback of these algorithms is their sensitivity towards weights or demand-levels. The decision maker must have a thorough knowledge of the priority of each objective before forming the single objective from a set of objectives. The obtained solution

depends on the underlying weight-vector or demand-level. Thus, for different situations, different weight-vectors need to be used and the same problem needs to be solved a number of times. This aspect is illustrated by considering a simple example.

A simple two-objective problem $F1$ of one variable is considered to illustrate the concept of multiple Pareto-optimality. This problem was used for the same purpose by Vincent and Grantham (1981) and subsequently by Schaffer (1984). The problem has two objectives and is shown in figure 2.1 and figure 2.2 :

$$\begin{aligned} \text{Minimize } f_{11} &= x^2, \\ \text{Minimize } f_{12} &= (x-2)^2. \end{aligned} \tag{2.7}$$

From the plot showing the performance space, it is clear that the Pareto-optimal solutions constitute all x varying from 0 to 2. The solution $x = 0$ is optimum with respect to f_{11} but not so good with respect to f_{12} and the solution $x = 2$ is optimum with respect to function f_{12} and not so good with respect to f_{11}. Any other point in between is a compromise to the above two functions and is a Pareto-optimum point. But the solution $x = 3$, for example, is not a Pareto-optimum point since this point is not better than the solution $x = 2$ with respect to either function. Among the Pareto-optimal points, the decision maker may want to prefer one point over the other depending on the demanding situation. but before taking any decision, he or she may want to know the possible Pareto-optimal solutions. The traditional methods cannot find multiple Pareto-optimal solutions simultaneously. For example, with all above methods and with equal priority to both functions having a weight vector (0.5,0.5), and demand-levels as individual optima, the obtained solution is $x^* = 1$. A weight vector (1,0) results in a scalarized objective as f_{11}. This gives a solution which is very good in f_{11} but not so in f_{12}, or the solution $x^* = 0$. Similarly weight vector (0,1) produce minimum of f_{12} or $x^* = 2$. Any point in the range $0 \leq x \leq 2$ may be a valid compromise and can be obtained with a particular choice of a weight vector. Thus, in order to obtain a particular solution the user has to know the corresponding weight vector, which is a difficult problem by itself. Another problem of using classical

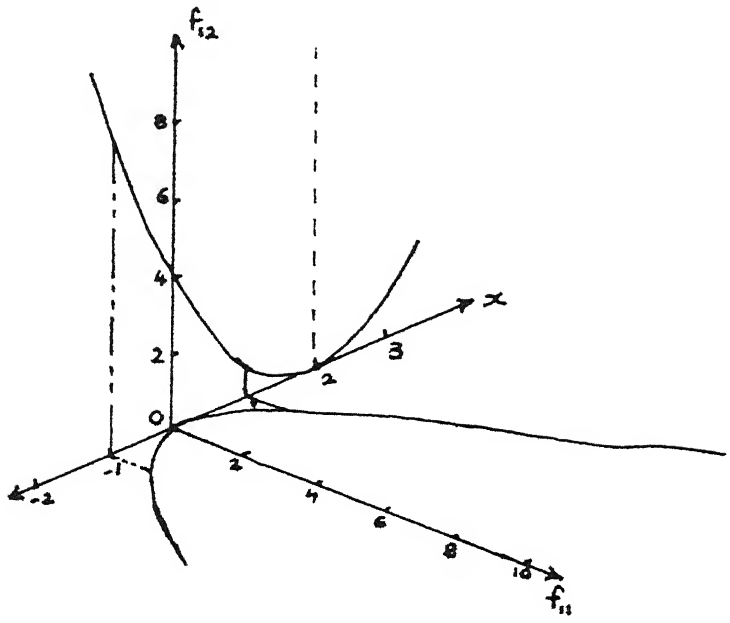


Figure 2.1: Functions f_{11} and f_{12} are plotted versus x.

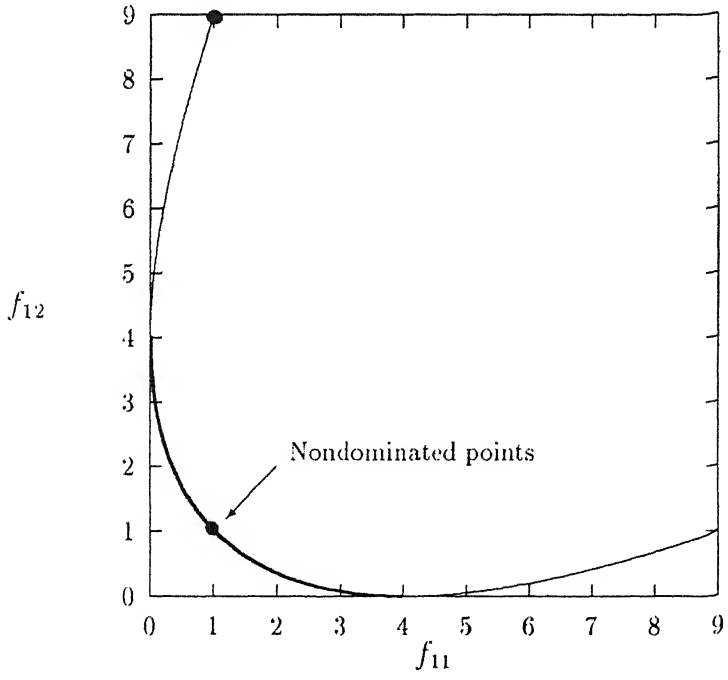


Figure 2.2: The performance space of problem F1 is shown.

methods is that oftentimes some objectives may involve uncertainities. If the objective functions are not deterministic, the fixation of a weight vector or a demand-level may become even more difficult. This discussion suggests that the classical methods to handle multiobjective optimization problems are inadequate and inconvenient to use. A more realistic method would be one that can find multiple Pareto-optimal solutions simultaneously so that users or decision makers may be able to choose the most appropriate solution for the current situation. Real-world and major decisions require higher-level (managerial or higher) intervention. Designers have no right to decide whether to build a car like Maruti or Ambassador. It is best for the designers to present as many compromise solutions as possible to management and help them decide. The knowledge of many Pareto-optimal solutions is also useful for later use, particularly when the current situation has changed and a new solution is required to be implemented. Since genetic algorithms deal with a population of points instead of one point, multiple Pareto-optimal solutions can be captured in the population in a single run.

2.5 Summary

In a multiobjective optimization problem, solution exists in the form of a set of points known as Pareto-optimal solutions or nondominated points. A designer or decision maker will be interested in a set of solutions, instead of a single point. In trying to solve these problems, most traditional methods scalarize the objective vector in to a single objective. This always results in a single point solution to the problem. They also require information about individual optima of all objectives, prior to the scalarization. Another serious disadvantage is that, methods like distance functions do not guarantee a Pareto-optimum. Since genetic algorithms work with a population of points and posses implicit parallelism, they show some promise in finding multiple Pareto-optimal solutions quickly in a single run.

The following chapter introduces genetic algorithms in function optimization.

Subsequent chapters describe modifications to this simple genetic algorithms to solve multiobjective optimization problems.

Chapter 3

Introduction to Genetic Algorithms

This chapter presents the fundamentals of genetic algorithms in function optimization. Genetic algorithms (GAs) are artificial search techniques based on natural law of population genetics. According to Goldberg (1989), they combine the Darwinian survival of the fittest and a stochastic information exchange procedure to form a robust search technique. The following section presents the mechanics of genetic algorithms.

3.1 Mechanics of Genetic Algorithms

In an artificial genetic search the coordinates of the search space (also known as parameter set) are coded into a finite length of string. A point in the space is represented by a string and the decoded values of the string's contents will correspond to its coordinates. This type of coding enables GA to simulate the characteristics of natural chromosome. Usually, in most cases, binary coding is adopted, even though this is not absolutely necessary. These coded strings are similar to the *chromosomes* in biological systems. In natural genetics the *genes* in chromosomes affect the physical qualities of an individual and also influence the characteristics of an offspring.

The gene values, known as *alleles*, completely describe an organism of an individual. Similarly in GAs, the binary string's '1' or '0' values describe a point's coordinates. In GA terminology, coded strings are called *chromosomes* and each point an *individual*. The chromosome's '1' or '0' value is known as an *allele*. An individual's objective function value is known as *fitness value* or simply *fitness*.

GAs are population based search techniques. They process a number of points, collectively known as *population*. In many GAs, the population size is kept constant. Stochastic methods are used to generate unbiased initial population in the search space. Unlike the traditional methods, which apply deterministic transition rules to update a point, GAs utilize probabilistic transition rules in each iterative step. Such discrete time step required to process and update an old population is known as *generation*. In each generation the population is processed by three main operators, which emulate the selection, mating and mutation of natural genetics. A GA which is composed of these three operators is popularly known as simple genetic algorithm, SGA. SGA's three important operators are :

- Reproduction,
- Crossover, and
- Mutation.

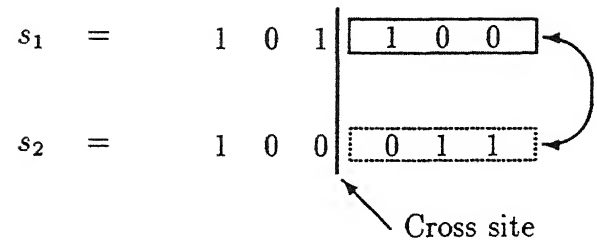
Reproduction or selection is the first operator in a GA. This operator makes copies of better individuals present in the population. Such individuals are identified by their fitness values. For example, in a maximization problem, good points are those points which have a fitness value above the average fitness of population and the best among them is the one which has the highest fitness. This selection process allows high fitness individuals to survive and reproduce, and low fitness strings to die. This is nothing but a simulated version of Darwinian survival of the fittest. Reproduction is the only operator in GA that processes individuals according to their fitness values.

Several selection schemes are in use today, and a detailed comparative analysis is made in Goldberg and Deb (1991). One method of selection is *proportionate selection*. This method simulates a spinning roulette wheel with slots sized according to the fitness values. In a maximization problem, using this method, the probability of being selected for copying is $f_i / \sum_j f_j$, where f_i is the fitness of string i, and $\sum_j f_j$ is the sum of all fitnesses. The expected number of copies of an i-th string is found by $f_i/\overline{f}$, where $\overline{f}$ is the average fitness of the population. A modified version of this method is known as *stochastic remainder proportionate selection*. In this method the number of copies are made according to the integer and fractional parts of the expected count $(f_i/\overline{f})$. This method minimizes the chance of random (blind) selection of some individuals, which may occur in a roulette wheel simulation. These methods can be applied only to maximization problems with positive fitness values. In order to minimize a function, the function has to be modified such that the maximum obtained will be the true minimum of actual function (duality principle). Several scaling methods have been suggested elsewhere (Goldberg, 1989) to handle negative function values.

Tournament selection is another selection process which can be applied to minimization as well as maximization problems without any restraint on the sign of fitness values. This method selects the best individual from a group of randomly chosen individuals. This process is carried until the mating pool is completely filled. The number of individuals considered for selection of an individual is known as *tournament size*. A tournament size of 2, known as binary tournament, is used in most of GA applications.

Crossover is the second operator in an SGA. This is a two stage operator. In the first stage, two strings are picked up at random from the newly reproduced strings in the mating pool and a cross site is selected, also at random, over the string length. In the second stage, two new strings (offsprings) are produced from the mating strings (parents), by swapping their string segments on one side of the cross site. For example,

in case of a two 6-bit binary coded strings, s_1 and s_2, the mating occurs as shown below.



The cross site is as shown. After mating the two new strings, s_1' and s_2', will be

s_1' = 1 0 1 0 1 1

s_2' = 1 0 0 1 0 0

The crossover operator is an artificial version of natural chromosome exchange in a genetic mating. Reproduction and crossover are the major operators in a GA. While the reproduction copies the best strings in the population, crossover produces new strings by exchanging some genetic material (1's or 0's in a binary string) between them. Together they search for new high-performance individuals. The power of genetic search lies in the selection of high-quality strings and the structural information exchange between them. The total number of strings participated in the mating can be controlled by specifying crossover probability, p_c. This parameter is the ratio of total number of strings selected for mating and the population size.

Mutation plays a secondary role in a GA. It helps to recover the loss of some potentially useful genetic material which might have been lost or absent in a genetic search. Mutation is the occasional random alteration of an allele's value (changing a 1 to a 0 and vice versa). The mutation probability, p_m, is kept very low as it disrupts

a string. Goldberg (1989) describes it as an insurance policy against premature loss of important notions. This process is illustrated below.

$$1\ \boxed{0}\ 1\ 0\ 1\ 1 \quad \xrightarrow{\text{Mutation}} \quad 1\ \underline{1}\ 1\ 0\ 1\ 1$$

A flow chart of a simple genetic algorithm is shown in figure 3.1. The next section presents some mathematical fundamentals of a genetic search.

3.2 Mathematical Fundamentals - The Schema Theorem

A GA actually discretizes the variable space by adopting a coding method. By processing certain strings which are similar in a way, it actually process a region. Consider as an example a one variable problem with 6-bit binary coding and the following strings in the population.

$$\begin{aligned} s_1 &= 1\ 1\ 0\ 1\ 0\ 1 \\ s_2 &= 0\ 0\ 1\ 1\ 0\ 0 \\ s_3 &= 1\ 0\ 0\ 0\ 1\ 1 \\ s_4 &= 0\ 1\ 1\ 1\ 1\ 0 \end{aligned}$$

There are several similarities among these strings. Holland (1975) defined a new entity, known as *schema* (*schemata*, plural), as a similarity template describing a subset of strings with similarities at certain string positions. The above strings have several schemata in common. By introducing a "don't care" symbol, '$*$', which can represent either a 1 or a 0, a schema can be expressed in terms of $\{0, 1, *\}$. Three schemata are shown below.

$$\begin{aligned} H_1 &= 1\ *\ 0\ *\ *\ 1 \\ H_2 &= 0\ *\ *\ *\ *\ * \\ H_3 &= *\ *\ *\ 1\ 0\ * \end{aligned}$$

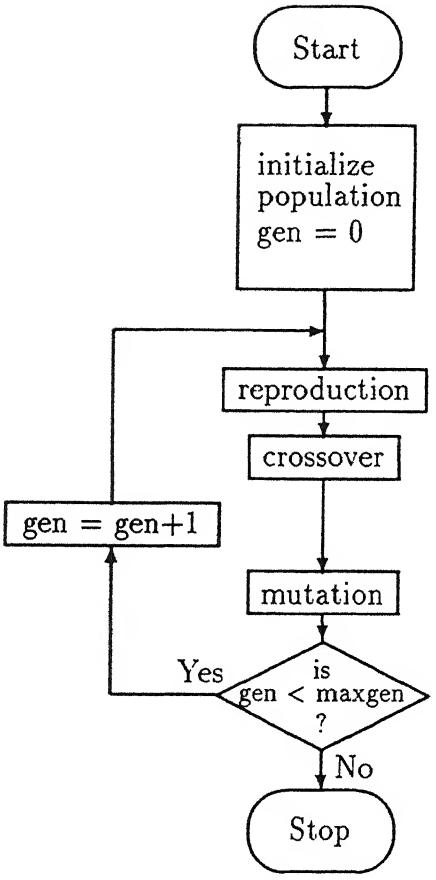


Figure 3.1: A flow chart of simple genetic algorithm, SGA.

Strings s_1 and s_3 belong to H_1, s_2 and s_4 belong to H_2, and s_1 and s_2 belong to H_3. In this example, total number of schemata available will be 3^6 while the total number of strings are 2^6 (string length is 6). One can see that a schema represents a region in the variable space. In this example, $(0 * * * * *)$ and $(1 * * * * *)$ represent two equal portions of entire variable space. Supposing the optimum doesn't lie in $(0 * * * * *)$ region, the strings belonging to this schema will not survive compared to the strings of $(1 * * * * *)$. The GA will soon recognize this fact and the population of strings belonging to $(1 * * * * *)$ schema will increase in coming generations. This effect can be explained, mathematically, by considering two important properties of a schema - *order* of a schema, $o(H)$, and its *defining length*, $\delta(H)$. The order of a schema is the number of fixed bits or positions (1's or 0's) present in the template and its defining length is defined as the distance between two extreme fixed bits. In case of the three schemata H_1, H_2 and H_3:

$$\begin{aligned} o(H_1) &= 3, o(H_2) = 1, o(H_3) = 2, \\ \delta(H_1) &= 5, \delta(H_2) = 0, \delta(H_3) = 1. \end{aligned}$$

Denoting the population of strings belonging to a schema H, at generation t, as $m(H,t)$, the fitness of the schema, $f(H)$, can be defined as :

$$f(H) = \frac{\sum_{s_i \in H} f(s_i)}{m(H,t)} .$$

During the selection process a schema will survive if its fitness is greater than the average fitness of population. In other words, the population of a schema will increase or decrease proportional to the factor $f(H)/\overline{f}$. The disruption of a schema during crossover depends upon its length, $\delta(H)$, and during mutation on its order, $o(H)$. Therefore, the expected number of strings of a particular schema at generation $t+1$ can be expressed as (Goldberg, 1989) :

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\overline{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - p_m o(H)\right] ,$$

where p_c and p_m are the crossover and mutation probabilities, and l is the string length. The above expression is a lower bound on the expected count of schema H. For a **short, low order, highly fit** schema, the quantity,

$$\frac{f(H)}{\overline{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - p_m o(H)\right] ,$$

will be always greater than 1. Therefore, such types of schemata will receive exponentially increasing number of copies in future generations. This theorem is known as *Schema Theorem* or The Fundamental Theorem of Genetic Algorithms. The schemata with above qualities are known as *building blocks*. By processing population of strings of size n, and allocating increasing number of copies to building blocks, GA implicitly process n^3 schemata without any special accounting or memory. Holland (1975) named this special ability of GA as *implicit parallelism*. As the generations proceed, the low order building blocks produce higher order, highly fit building blocks, eventually leading to optimal or near optimal strings.

3.3 Advantages of GAs

Most of the traditional algorithms are point based techniques and use deterministic transition rules to update a point. These can be classified as direct and descent methods (Rao, 1991). Direct search methods require objective function values only, and update a point using local exploration. Descent methods are gradient based techniques. They require first and often higher order differentials of objective function. Both these methods have a tendency to converge, prematurely, to a local peak. These methods are suitable only to a class of problems. For discontinuous or C^0 continuous functions gradients cannot be found, which limits their application. In view of these drawbacks, we can draw some important differences between GAs and their traditional counterparts. They differ in four ways (Goldberg, 1989) :

1. GAs work with a coded parameter set.

2. GAs are population based search techniques.

3. GAs require only the objective function values.

4. GAs use probabilistic transition rules in updating the population.

The coding in GAs finely discretizes the variable space. This type of discretizing enables GAs to handle discontinuous as well as continuous functions. This helps GAs to handle a wide variety of problems. The genetic search starts from a population of random points, uniformly distributed over the variable space. This property of population based search helps enhanced GAs (Deb, 1993) to present a set of solutions instead of a single point. By processing schemata, GAs implicitly conducts a parallel search, which makes it to find a global solution. Reproduction is the only operator in GAs that uses the objective function and it does not require any auxiliary information such as gradients. GAs are "blind", because they operate as a black box taking input in the form of objective function values and giving output in the form of highly fit individuals. The unbiased probabilistic rules can update a population, no matter how the function varies. This permits them to be applied to a wide class of problems with higher efficiency than their traditional counter parts. This ability, known as *robustness*, promises a bright future for GAs in function optimization.

3.4 Summary

Genetic search resembles the mechanics of natural genetics. GAs process a population of points which are coded as strings so as to simulate the characteristics of natural chromosomes. A simple GA is composed of three operators - reproduction, crossover, and mutation. Reproduction copies the best strings while crossover exchanges string segments of these copies. Together they search for new high-performance notions. Mutation is a secondary operator, which tries to recover the lost or absent genetic material by occasional alteration of an allele. By processing similar strings, GA

process a number of schemata, which represent regions in the variable space. It is argued that GAs allocate exponentially increasing number of copies to short, low order, highly fit schemata, known as building blocks. As generations proceed, these building blocks form higher order, high-quality schemata, which results in the narrowing of the search region. Finally, this process leads to an optimal or a near optimal string. GA's implicit parallelism, due to its stochastic operators and population of individuals, increases the probability of finding the global optimum. This special ability makes it a robust tool in solving a wide range of problems.

The next chapter presents an early extension to this SGA to solve multiobjective optimization problems.

Chapter 4

Vector Evaluated Genetic Algorithm

As early as in 1967, Rosenberg suggested, but did not simulate, a genetic search to the simulation of the genetics and chemistry of a population of single-celled organisms with multiple properties or objectives (Rosenberg, 1967). Rosenberg performed selection and mating according to an antifitness function, $f_i = \sum_j (x_j - \overline{x}_j)^2$, where f_i is the i-th properity, x_j is chemical concentration and $\overline{x}_j$ is desired concentration (*goal*). The sum is taken over all chemicals present in the i-th properity. The inverse of f_i values were considered in reproduction and mating. Rosenberg experimented with a single properity ($i = 1$) and suggested an extension of his work as an application to multiobjective problems.

First practical algorithm, called Vector Evaluated Genetic Algorithm (VEGA), was developed by J. D. Schaffer in 1984 (Schaffer, 1984). The following section presents this algorithm.

4.1 Schaffer's VEGA

Schaffer modified simple tripartite genetic algorithm by dividing the population into equal subpopulations for each objective and selecting from each pool separately according to each objective. He modified J. J. Grefenstette's GENESIS program (Schaffer, 1984) by creating a loop around traditional selection procedure, so that it was repeated for each individual reproduction. Then entire population is thoroughly shuffled to apply mating and crossover operators. This is performed to achieve the mating of individuals of different subpopulation groups. Schaffer also incorporated a procedure to identify the nondominated individuals in order to analyze the VEGA's performance. This algorithm is shown in figure 4.1.

One advantage of this algorithm is that, it insures the survival of any individual that performs above average in any objective. Schaffer's intention was to produce Pareto-optimal offsprings by mating above average strings of different pools. Although this procedure seems to be logical from a biological point of view, there is every possibility of a mating between strings of same pool. This cannot be avoided because of the random processes involved such as mate selection and shuffling. This algorithm worked efficiently for some generations but later suffered from a serious drawback which is its potential bias against middling or utopian individuals. The individual selection of specialists from each subpopulation resulted in speciation in the population. Outcome of this effect is the convergence of entire population towards individual optimum regions after a large number of generations. Being a decision maker, we may not like to have any bias against such utopian individuals, rather we may want to find any nondominated points. Schaffer tried to minimize this speciation by developing two heuristics — the nondominated selection heuristic (a wealth redistribution scheme), and the mate selection heuristic (a cross breeding scheme) (Schaffer, 1984). These two heuristics are briefly discussed in the following subsections.

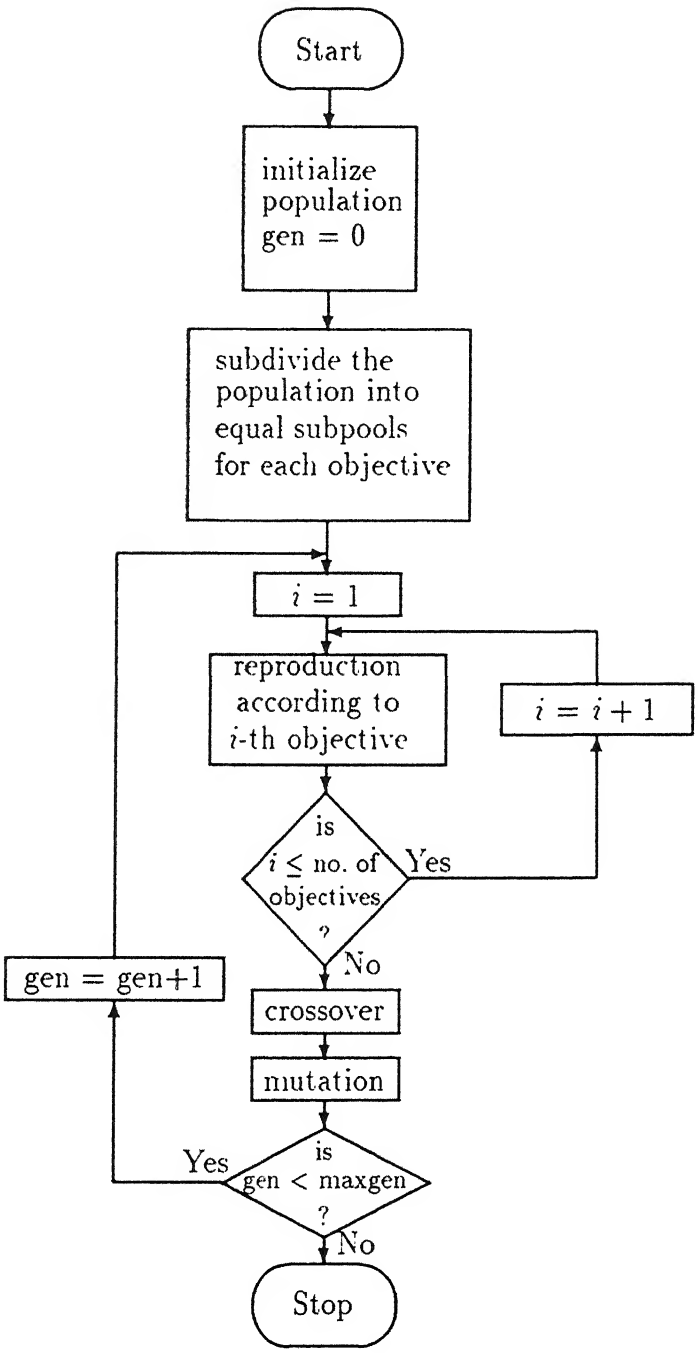


Figure 4.1: Schaffer's vector evaluated genetic algorithm, VEGA.

4.1.1 Nondominated Selection Heuristic

In this heuristic Schaffer attempted to give some preference for nondominated individuals during the selection process. He felt that by selecting high performance individuals from each pool, some points with good performance in all objectives, but without extremely good in any, might not survive. This resulted in speciation in the population. The nondominated selection heuristic was aimed at minimizing this speciation. In this heuristic, dominated individuals are penalized by subtracting a small fixed penalty from their scaled fitness. Then the sum of the penalties was divided among the nondominated individuals and was added to their fitness. But this algorithm failed when the population has very few nondominated individuals, resulting a large fitness value for those few nondominated points, eventually leading to a high selection pressure. For this reason, Schaffer had to drop this method though it was better than simple VEGA.

4.1.2 Mate Selection Heuristic

The mate selection heuristic was intended to promote the cross breeding of specialists from different subgroups. This was implemented by selecting an individual, as a mate to a randomly selected individual, which has the maximum Eucledean distance in the performance space from its mate. But it failed to prevent the participation of poorer individuals in the mate selection. This is because of random selection of the first mate and the possibility of a large Euclidean distance between a champion and a mediocre. A modified version of mate selection was also considered using a measure known as "improvement distance" instead of Eucledean distance. Schaffer defined this distance as the sum of all positive differences between a mate's objectives and its proposed mate's objectives (for minimization only). The first mate is selected, as usual, from the mating pool at random. Schaffer found this method better than the first version, but still worse than the simple VEGA without it. Schaffer concluded that the random mate selection is far superior than this heuristic.

4.2 Simulation Results of Problem F1

A simple VEGA is implemented in 'C' language. This code is used to test VEGA's ability in finding nondominated points. This program is used to solve test problem F1, introduced in chapter 2. The same problem was also used by Schaffer to test the capabilities of VEGA. The results obtained, as a part of this thesis, are found to be similar to Schaffer's results (Schaffer, 1984).

In all simulations, the VEGA parameters used in the experiments are as follows:

Maximum generation	:	500
Population size	:	100
String length (binary code)	:	32
Probability of crossover	:	1.0
Probability of mutation	:	0.0
Tournament size	:	2

The parameters are held constant across all runs. To investigate the effect of VEGA alone, mutation probability is kept zero. Unbiased initial population is generated randomly spreading over entire variable space in consideration. Initial range for the design variable, x, used in simulations is $[-10, 10]$, but the nondominated region is only $[0, 2]$. The population drift is shown in figures 4.2 through 4.5. These figures are drawn in the performance space.

The initial population is shown in figure 4.2. At generation 10, VEGA's population moved towards Pareto-optimal front (figure 4.2). But at generation 100, some subpools are formed in the population. These clusters moved further towards individual optima with generation number. At generation 500, VEGA's population completely converged to three sub-regions which are close to individual optima (figure 4.5). These subpools are nothing but species formed in the population. These

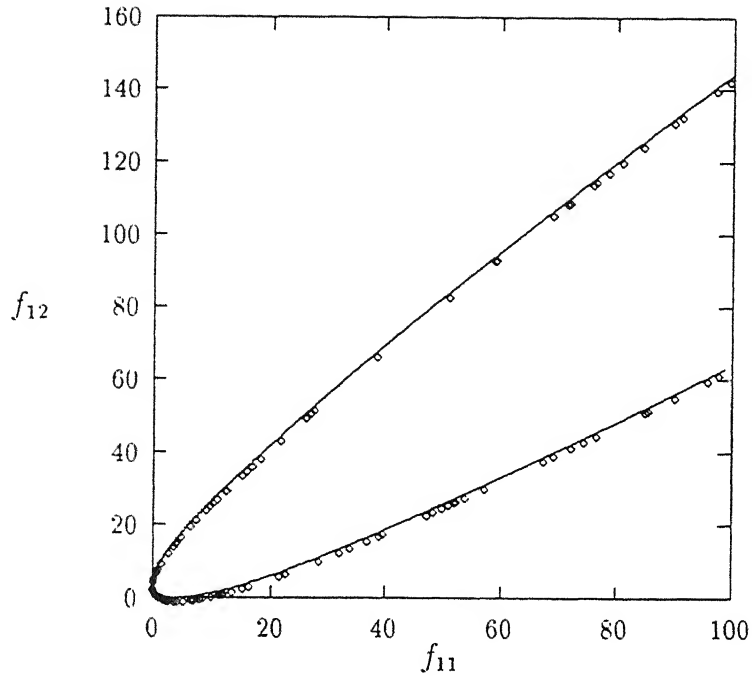


Figure 4.2: Population at generation 0 obtained using VEGA for problem F1.

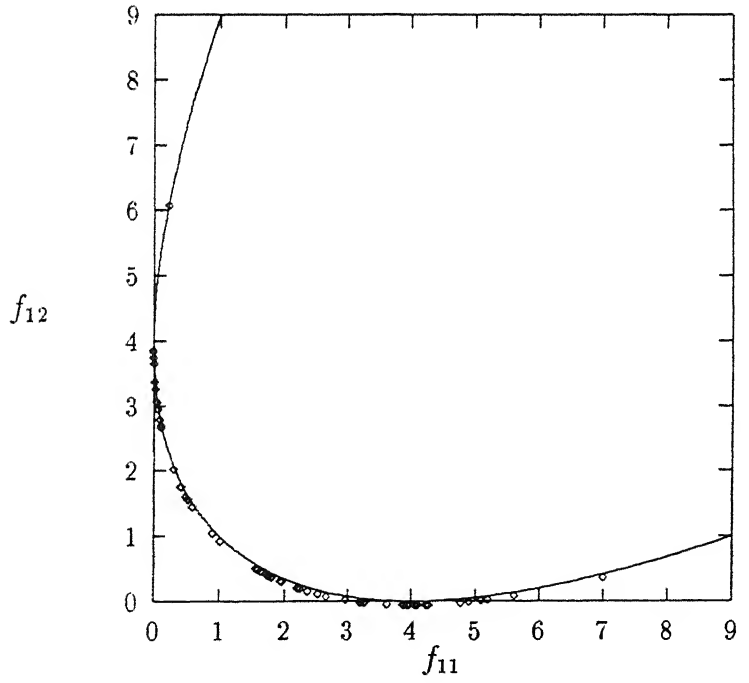


Figure 4.3: Population at generation 10 obtained using VEGA for problem F1.

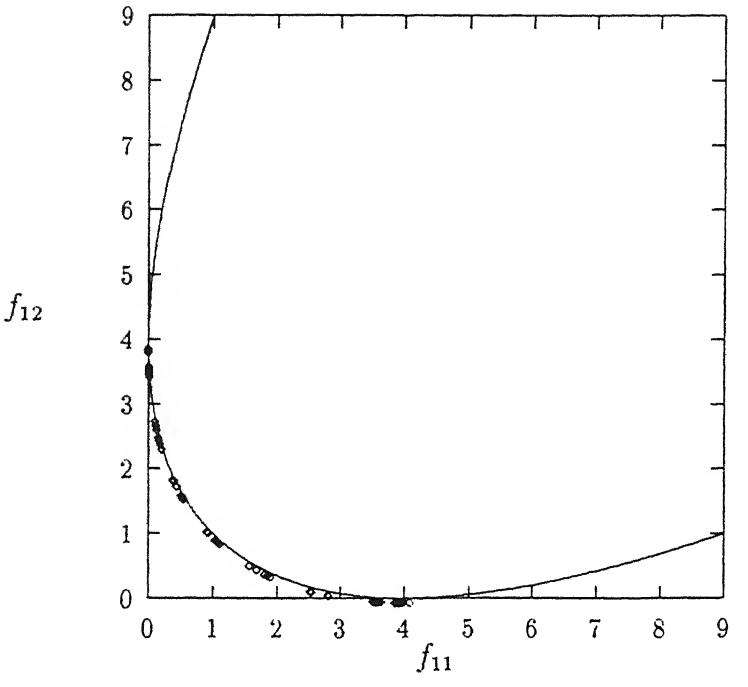


Figure 4.4: Population at generation 100 obtained using VEGA for problem F1.

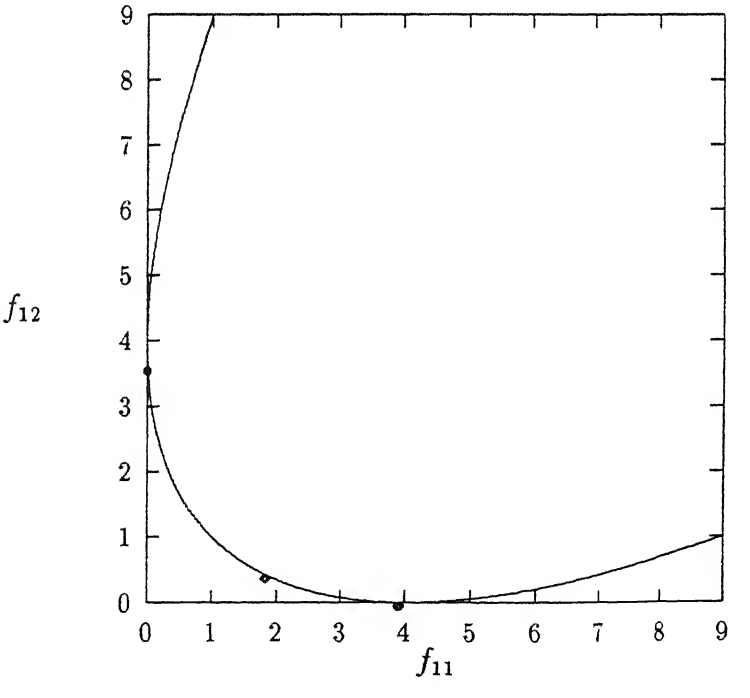


Figure 4.5: Population at generation 500 obtained using VEGA for problem F1.

results clearly show the VEGA's failure in maintaining stable populations at all nondominated points. This experiment reiterates VEGA's lack of control over the speciation in the population. One method to increase uniform distribution of points is through a nondominated sorting procedure in conjunction with a sharing technique, as suggested by Goldberg (1989). That algorithm is discussed in the following chapter.

4.3 Summary

Schaffer's VEGA was based on the principle that mating between two best strings with respect to different objectives, might produce a compromising offspring. To implement this, he modified SGA by dividing the population into equal subpools, as many as the number of objectives, and selecting from each pool according to an objective. This algorithm worked efficiently for some generations but later suffered from a serious drawback which is its potential bias against middling or utopian individuals. As a result, the population tended to drift towards individual optima of objectives. To minimize this drift, Schaffer considered two heuristics. The first one, he called nondominated selection heuristic - a wealth redistribution scheme - was dropped as it offered the possibility of a high selection pressure. Second method, the mate selection heuristic, was intended to promote the cross breeding of above average individuals of different pools. Schaffer's experiments revealed that simple VEGA's random mate selection was far better than this heuristic. Simulation results of problem F1, have showed that VEGA's population distribution is biased towards individual optima. This result also show VEGA's inability in distributing its population over Pareto-optimal region.

VEGA's failure in keeping stable subpopulations at some middling regions prompted researchers to come up with new suggestions. One such suggestion, by David Goldberg, is pursued in the next chapter.

Chapter 5

Nondominated Sorting

The idea behind the nondominated sorting procedure is that a ranking selection method is used to emphasize good points and a niche method is used to maintain stable subpopulations of good points. This algorithm is developed based on this concept. Since the algorithm is based on nondominated sorting procedure, this algorithm is named as nondominated sorting genetic algorithm, NSGA.

5.1 Nondominated Sorting Genetic Algorithm

NSGA varies from simple genetic algorithm only in the way the selection operator works. The crossover and mutation operators remain as usual. Before the selection is performed, the population is ranked on the basis of nondomination described in chapter 2. The nondominated individuals present in the population are first identified from the current population. All these individuals are assumed to contribute the first nondominated front in the population and assigned a dummy fitness value proportional to the population size. This dummy fitness is intended to give equal reproductive potential for all these nondominated individuals. In order to maintain diversity in the population, these classified individuals are shared with their dummy fitness values (fitness sharing is a way of finding and maintaining multiple optimal

Table 5.1: An illustration of nondominated sorting procedure.

Individual	x	Fitness 1 x^2	Fitness 2 $(x-2)^2$	Rank / Front	Dummy fitness before sharing	Dummy fitness after sharing
1	-1.5	2.25	12.25	2	3.00	3.00
2	0.7	0.49	1.69	1	6.00	6.00
3	4.2	17.64	4.84	2	3.00	3.00
4	2.0	4.00	0.00	1	6.00	3.43
5	1.75	3.06	0.062	1	6.00	3.43
6	-3.0	9.00	25.0	3	2.00	2.00

solutions in a simple GA (Goldberg and Richardson, 1987; Deb, 1989). Then these individuals are ignored temporarily to process the rest of population in the same way to identify individuals for the second front. These new set of points are then assigned a new dummy fitness value which is kept smaller than the minimum shared dummy fitness of the previous front. This process is continued until the entire population is classified into several fronts. To illustrate the sorting procedure better, the problem F1 is considered again. For example, consider a population of 6 points shown in table 5.1. Each individual has two fitness values. First fitness corresponds to the first objective, $f_{11} = x^2$, and the second fitness to the second objective, $f_{12} = (x-2)^2$. Using equation 2.2, these individuals are ranked as shown. Since both fitness values, f_{11} and f_{12}, of the second individual are less than that of the first individual, second individual dominates the first individual. This way first, third, and sixth individuals are classified as dominated points.

This sorting will produce the first batch of nondominated points, second, fourth,

and fifth individuals, as they do not dominate each other. These individuals belong to the first front and assigned equal dummy fitness value, say 6.0. Once the first front is found, all points in this front are ignored temporarily, to process the rest of the population in the same way. Thus we are left with first, third, and sixth individuals. Among these points, first and third individuals, classify the second front and sixth point classify the third front. In each front, dummy fitness is assigned in a way so that a latter front is inferior to a former front. For example, to first and third individuals, a fitness of 3.00 each and to sixth a fitness of 2.00 may be assigned. In actual practice, the dummy fitness values depend on the *sharing* of the individuals, an aspect which is discussed in the next section. These newly assigned values are kept slightly less than the minimum shared dummy fitness of the previous front. This process helps to set priorities on different individuals.

The population is then reproduced according to dummy fitness values. A stochastic remainder proportionate selection is used in this study. Since individuals in the first front has the maximum fitness value, they always get more copies than the rest of population. This was intended to search for nondominated regions or Pareto-optimal fronts. This results in quick convergence of the population towards nondominated regions, and sharing helps to distribute it over this region. In reality, the exact nondominated region is the one that is obtained after comparing all the points in feasible domain. If a point is locally dominated, it is also globally dominated. But the converse is not true. In each generation, NSGA finds locally nondominated points only. However, if there exists any isolated globally nondominated point NSGA insures its survival. This is because any such individual is likely to have highest possible dummy fitness. The power of NSGA lies in the successful transformation of a multiobjective problem, no matter how many objectives are there, to a single function problem without losing the concept of vector optimization. By selecting nondominated points, NSGA is actually processing those schemata that represent Pareto-optimal regions.

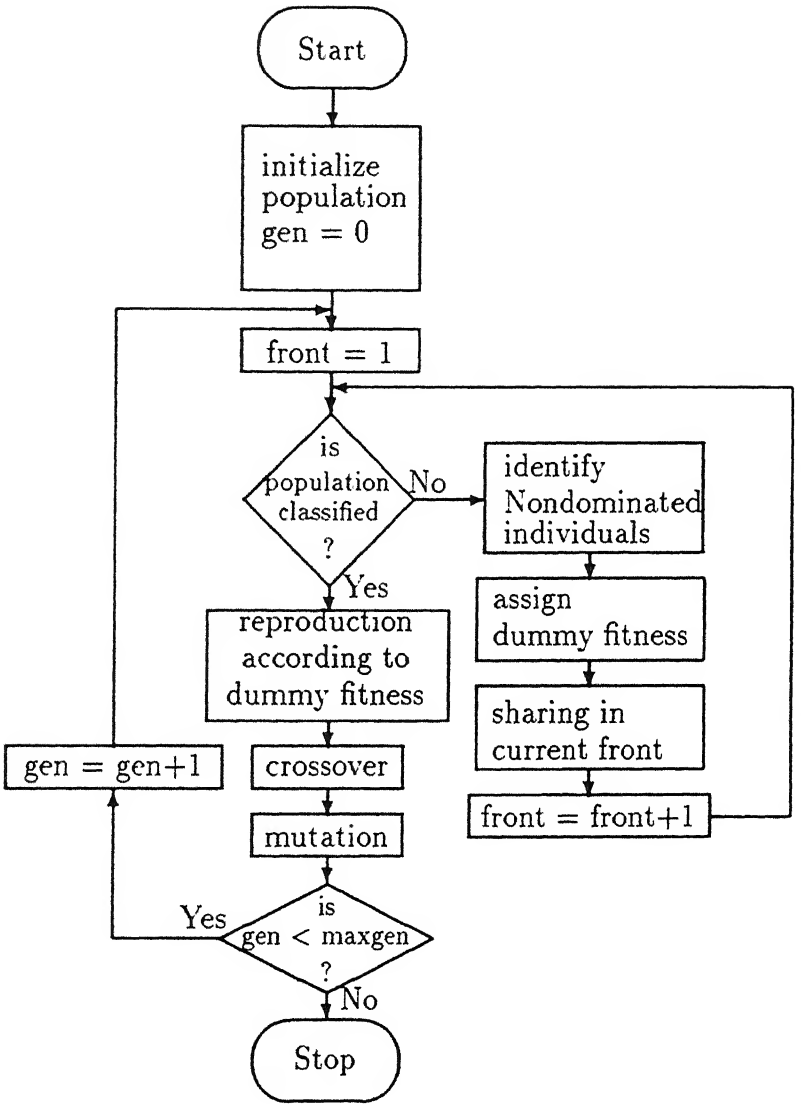


Figure 5.1: Nondominated sorting genetic algorithm, NSGA.

Therefore, the building blocks for NSGA will be those schemata that represent globally nondominated individuals. Figure 5.1 shows a flow chart of this algorithm. It is interesting to note that both minimization and maximization problems can be handled by this algorithm. It requires a simple changing of selection process of nondominated individuals as stated in chapter 2.

5.2 Fitness Sharing

In dealing with multimodal problems, simple GAs converge to one optimum and cannot converge to many alternatives. Whether GAs will converge to this or that optimum is usually unpredictable. This problem with finite populations is known as the *genetic drift*. Since every nondominated individual in NSGA will be assigned equal dummy fitness, convergence of population to a solution is completely unpredictable. A simple nondominated sorting procedure cannot control this type of genetic drift. In fact, population must be distributed in order to find as many Pareto-optimal points as possible. Simple GAs are not robust enough to find multiple peaks in a single run. But GAs can be used to form stable subpopulations of individuals by artificially forming niches corresponding to each optimum.

In nature, niche can be viewed as an organism's job or role in an environment and species as a group of organisms with common characteristics. In artificial genetic search multiple peaks or function subdomains correspond to niches, and strings serving those domains to species. By artificially forming peaks in the search space, strings can be allocated to each peak proportional to its magnitude.

Goldberg and Richardson (1987) developed a niching technique by defining a function $Sh(d_{ij})$ as a function of distance metric (d_{ij}) between i-th and j-th individuals as:

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{\text{share}}}\right)^{\alpha}, & \text{if } d_{ij} < \sigma_{\text{share}}; \\ 0\ , & \text{otherwise.} \end{cases} \tag{5.1}$$

The purpose of sharing is to degrade an individual's fitness value by dividing it by

the accumulated number of shares as follows:

$$f_s(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{j=1}^{P} Sh(d_{ij})}, \tag{5.2}$$

where P is the population size. If there are more number of individuals around an alternative, the fitness of that individual (and those around it) will be degraded more. This helps to keep diversity in the population and delay convergence of individuals to a particular alternative. The sharing parameter σ_{share} is used to control the extent of sharing. When the distance-metric (d_{ij}) is measured in decoded parametric space, it is called *phenotypic sharing*. The calculation of the distance-metric and corresponding sharing parameter value is described elsewhere (Deb, 1989).

For a p-dimensional decision-vector, the distance-metric is calculated as follows:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} (x_{k,i} - x_{k,j})^2}, \tag{5.3}$$

where $\mathbf{x}_i = \{x_{1,i}, x_{2,i}, \ldots, x_{p,i}\}^T$ and $\mathbf{x}_j = \{x_{1,j}, x_{2,j}, \ldots, x_{p,j}\}^T$ are decoded vectors of i-th and j-th strings. To induce q number of niches over entire parametric space, the σ_{share} parameter is calculated from the following equation (Deb, 1989;Deb and Goldberg, 1989)

$$\sigma_{\text{share}} = \frac{\sqrt{\sum_{k=1}^{M} (x_{k,max} - x_{k,min})^2}}{2q^{1/p}}. \tag{5.4}$$

The above equation has been formed assuming each niche as a p-dimensional hypersphere of radius σ_{share}, such that each sphere encloses $\frac{1}{q}$ of the hypervolume of the whole space. Equations 5.3 and 5.4 are used in NSGA to calculate d_{ij} and σ_{share}. In case of parameter set with unequal bounds, a normalized distance metric can be

$$d_{ij} = \sqrt{\sum_{k=1}^{p} ((x_{k,i} - x_{k,j})/(x_{k,max} - x_{k,min}))^2}, \tag{5.5}$$

and σ_{share} corresponding to this normalized distance metric will be

$$\sigma_{\text{share}} = \frac{1}{2q^{1/p}}. \tag{5.6}$$

Consider the example population shown in table 5.1. If $\sigma_{\text{share}} = 1.0$ is chosen, then second individual do not share with fourth or fifth individuals, because the absolute difference in decoded parameter space between second and fourth & fifth are not less than 1.0. Thus $\sum_{j=2,4,5} Sh(d_{2j}) = 1.0$ and $f_s(x_2) = 6/1 = 6.0$. On the other hand, fourth individual shares with fifth individual and $Sh(d_{45}) = \left(1 - \frac{2-1.75}{1.0}\right) = 0.75$. Thus shared fitness of fourth individual is $f_s(x_4) = 6/(1 + 0.75) = 3.429$. Similarly, the shared fitness for fifth individual is $f_s(x_5) = 6/(1 + 0.75) = 3.429$. Since, the minimum shared fitness in the first front is 3.429, we assign dummy fitness values of all individuals in the second front equal to 3.0 (say) . Clearly, the two individuals in this front do not share with each other. Thus their shared fitness is same as their dummy fitness values : $f_s(x_1) = f_s(x_3) = 3.0$. Since. there is only one individual in the third front, the shared fitness is $f_s(x_6) = 2.0$.

Individuals in each front are shared separately so as to maintain diversity of solutions in each front. Since individuals in the first front has greater dummy fitness value, the population quickly converges towards the first front. Then sharing controls number of individuals around a point, thereby distributing the population over the entire front. This helps to avoid convergence towards individual champions. The degree of distribution can be controlled by σ_{share} parameter, which depends on the desired number of Pareto-optimal points.

The distribution powers of NSGA can be tested by considering the problem F1. The following section presents the simulation results of F1 obtained using NSGA.

5.3 Simulation results of Problem F1

This problem was introduced in chapter 2, and VEGA's results are presented in chapter 4. In all simulations for this problem, the NSGA parameters used in the experiments are as follows:

Maximum generation	:	500
Population size	:	100
String length (binary code)	:	32
Probability of crossover	:	1.0
Probability of mutation	:	0.0
σ_{share}	:	0.1

The parameters are held constant across all runs. To make the comparison fair, the initial population used for NSGA is same as that generated for VEGA. This is shown in figure 5.2. The population drift is shown in figures 5.2 through 5.5, drawn in the performance space.

As mentioned earlier, the initial population (at generation 0) is exactly same for both VEGA and NSGA. At generation 10, like VEGA, NSGA's population converged towards the nondominated region. At generation 100 (figure 5.4), the difference in distribution is clear, NSGA's population is distributed uniformly, while VEGA formed some subpools of strings (figure 4.4). Unlike VEGA, NSGA continued to maintain this distributed population. This can be observed in figure 5.5,which shows population at generation 500. This figure also shows the ability of NSGA in distributing the population uniformly and maintaining it till generation 500.

In view of these results, some comparisons can be drawn between VEGA and NSGA. These are discussed in the following section.

5.4 Comparisons between VEGA and NSGA

VEGA's search is not directed towards nondominated regions. This is because its selection is biased towards individual optima. In other words, it is incapable of recognizing the building blocks required for multiobjective optimization. In case of NSGA, the nondominated individuals are identified before the selection process

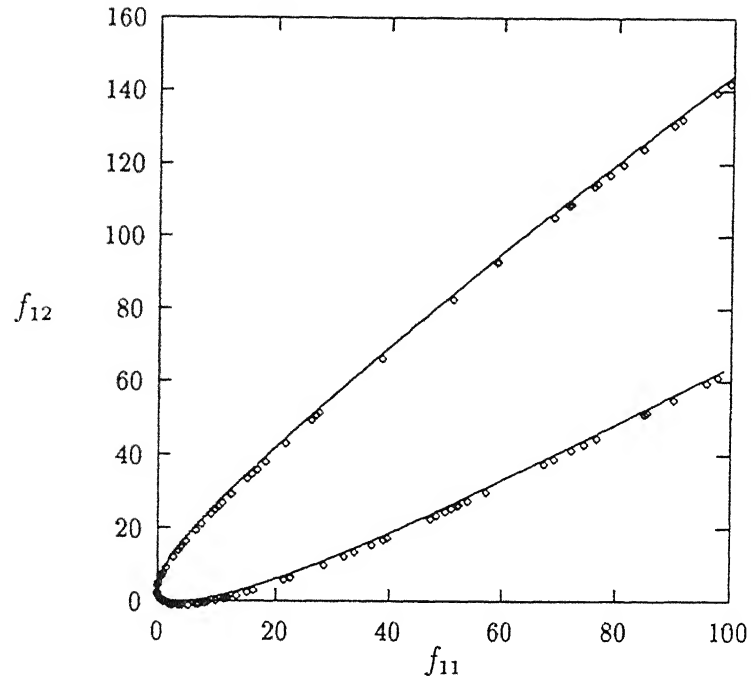


Figure 5.2: Population at generation 0 obtained using NSGA for problem F1.

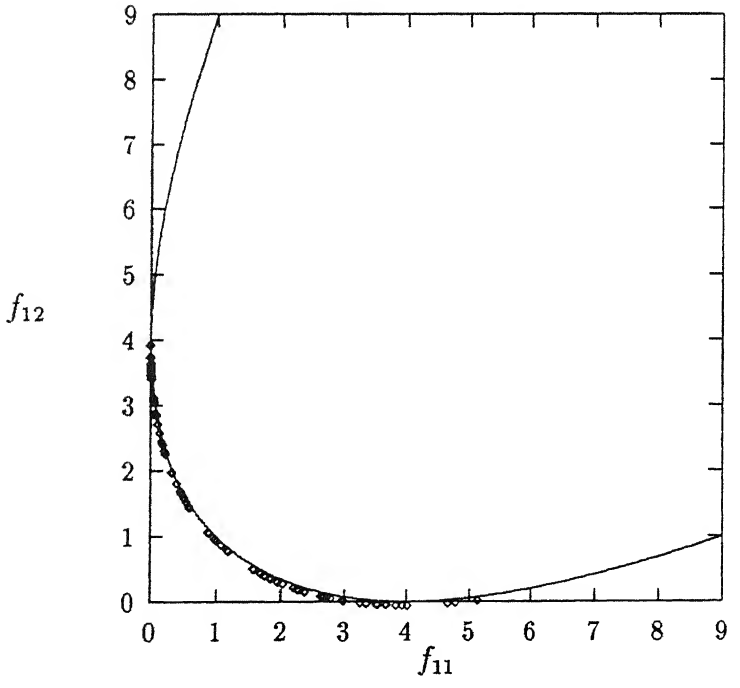


Figure 5.3: Population at generation 10 obtained using NSGA for problem F1.

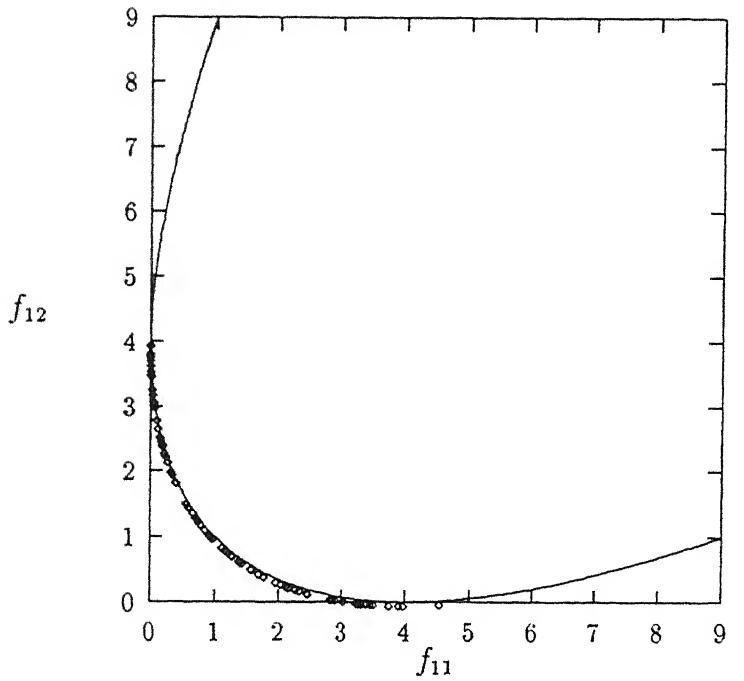


Figure 5.4: Population at generation 100 obtained using NSGA for problem F1.

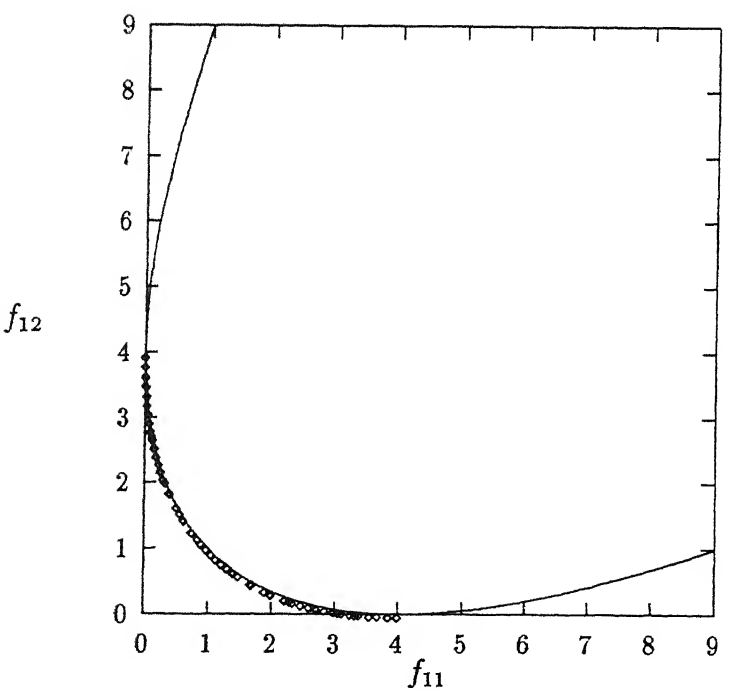


Figure 5.5: Population at generation 500 obtained using NSGA for problem F1.

begins. Also, NSGA ranks all individuals on the basis of nondomination. This is important, because even the dominated points can be ranked (classified as fronts) using this concept. This process enables NSGA to set preferences on respective strings of different fronts. It is easy to visualize that in every classified front there will always be one or more than one individual. Every member of such group must have equal chance of reproducing (survival probability). In fact, the inability to provide such equal priority is the main reason for VEGA's failure. At the same time, NSGA assigns equal dummy fitness to all such members, giving them equal reproductive strength. This process not only helps to avoid the bias present in VEGA, but also enables to process the required building blocks. For example, consider the problem F1 (figures 2.1 and 2.2) presented in chapter 2. In this problem the nondominated or Pareto-optimal region is $0 \leq x \leq 2$. Assuming the variable span as $-8 \leq x \leq 8$ and a 4-bit binary coding in the simulation, the nondominated region is represented by strings varying from 1 0 0 0 to 1 0 1 0 (approximately). Thus, this region is represented by schema (1 0 $*$ $*$). This becomes the building block for NSGA. Some times the initial population may not contain any true nondominated points. But, such situations also occur in single function optimization, where SGA's initial population may not have any representation of a building block. Crossover helps in these situations. by creating new strings that are likely to represent Pareto-optimal regions. This way NSGA searches for nondominated regions.

Another drawback of VEGA is its lack of control over speciation. Schaffer's two heuristics, nondominated selection and mate selection heuristics, failed to produce desired effects. NSGA employs a fitness sharing method, a niche and speciation technique, to control its population in forming subpools in the search space. It uses phenotypic sharing. which discourages many occupants in a species by degrading their dummy fitness values. Another disadvantage of VEGA is its population size. As the number of objectives increase, the population size has to be increased to minimize the bias of individual selection. NSGA is not affected by the population size and the

number of objectives. Only the string length and the required precision effect the size of population.

Other advantages of NSGA include its flexibility in handling multiobjective problems of minimization or maximization or both. Sharing does not effect the the actual function values as it changes dummy fitness values only. This way actual function values are preserved. Using proper fitness scaling methods on dummy fitness values selection pressure can be controlled. This may expedite NSGA's convergence towards Pareto-optimal solutions.

5.5 Summary

Nondominated sorting is the process of identifying nondominated individuals in the population and setting priorities on them. NSGA employs this procedure and a fitness sharing method to solve multiobjective optimization problems. In each generation, NSGA identifies nondominated points in groups. First group individuals are assigned a dummy fitness value proportional to the population size. Then these individuals are shared over dummy fitness. The succeeding groups are identified, in the same way, temporarily ignoring the previously processed groups. They will receive a dummy fitness value which is less than the minimum shared dummy fitness of its previous group. After assigning this fitness, each group is shared before sorting the next group. After the classification of entire population in this way, strings are copied according to their dummy fitness values. The rest of the algorithm, crossover and mutation, is similar to that of a simple GA. The power of NSGA lies in the identification of nondominated points and setting preferences on them. This helps NSGA to process necessary building blocks which represent Pareto-optimal regions. The major differences between VEGA and NSGA are the selection process and the niche and speciation technique. VEGA's selection is biased and Schaffer's heuristics failed to control the speciation. NSGA overcame both the difficulties using nondominated sorting and fitness sharing technique.

The next chapter presents the simulation results of NSGA and VEGA on two more test functions.

Chapter 6

Comparative Analysis of NSGA and VEGA

In this chapter, NSGA and VEGA are applied on three test problems previously used by Schaffer (1984) and others (Vincent and Grantham, 1981; Chankong and Haimes, 1983). The comparison is made based on the population movement and a chi-square-like distribution measure.

6.1 Test problems

The first two problems considered are of two single variable objectives and the third problem is a set of two two-parameter functions. Although some functions, other than the three functions, are used to validate the code, they are not discussed in this thesis.

Problem F1 : Two smooth, single-variable, unimodal objectives:

This problem was already presented in chapter 2 and its simulation results are discussed in chapters 4 and 5. This section briefly recapitulates the characteristics of

this test problem.

This multiobjective problem has two smooth unimodal functions, f_{11} and f_{12}, as shown in figure 2.1

$$\begin{aligned} \text{Minimize} \quad f_{11} &= x^2 , \\ \text{Minimize} \quad f_{12} &= (x-2)^2 . \end{aligned} \tag{6.1}$$

The figure 2.1 is drawn in variable space. The Pareto-optimal region can be found by exhaustive search and using equation 2.2. The points $x_{11}^* = 0$ and $x_{12}^* = 2$ are the respective minima of f_{11} and f_{12}. The region in between ($0 \leq x \leq 2$) is the Pareto-optimal region. This region is shown in performance space in figure 2.2. The equation of this front can be found by eliminating the variable x from the equations 6.1. This front is a smooth second order curve. This function was first used by Vincent and Grantham (1981) to illustrate the concept of Pareto-optimality. Later Schaffer (1984) used it as a preliminary test function for VEGA.

Problem F2 : A C^0-continuous bimodal objective and a smooth unimodal objective:

This problem consists of two objectives, f_{21} and f_{22}, which are :

$$\begin{aligned} \text{Minimize} \quad f_{21} &= -x && \text{if } x \leq 1 \\ &= -2 + x && \text{if } 1 < x \leq 3 \\ &= 4 - x && \text{if } 3 < x \leq 4 \\ &= -4 + x && \text{if } x > 4, \text{ and} \\ \text{Minimize} \quad f_{22} &= (x-5)^2 && . \end{aligned} \tag{6.2}$$

The first function, f_{12}, is a C^0-continuous function with minima at $x = 1$ and $x = 4$. Also the differentials of this function at $x = 1, 3$, and 4 do not exist. The second objective, f_{22}, is a parabola with a minimum at $x = 5$. these two functions are shown in variable space in figure 6.1 and in performance space in figure 6.2.

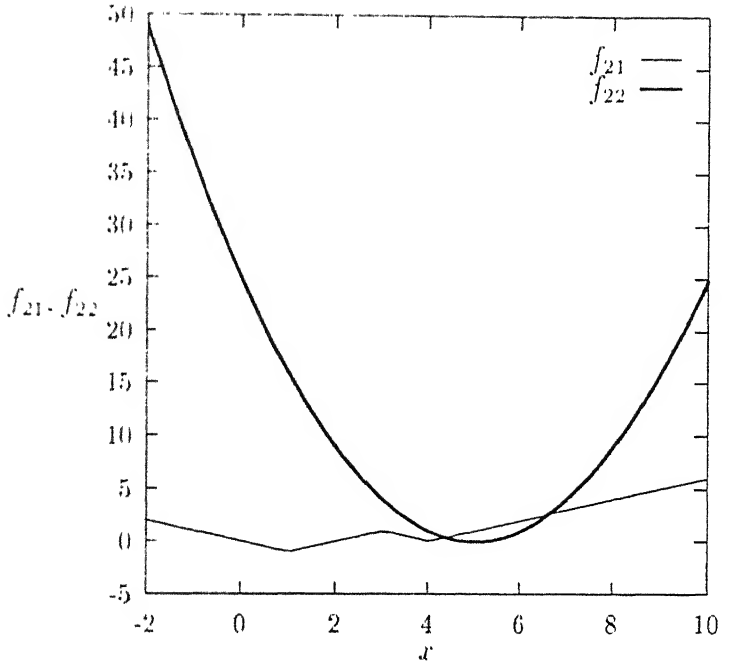


Figure 6.1: Problem F2 is plotted between f_{21},f_{22} and x.

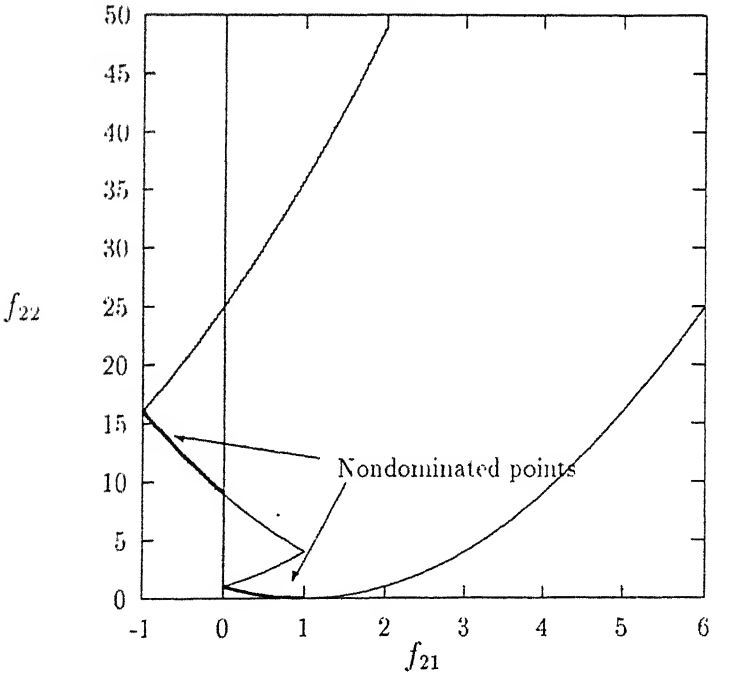


Figure 6.2: Problem F2 is plotted between f_{21} and f_{22}.

The speciality of this problem is its disjointed nondominated regions. These can be seen in figure 6.2 as regions $1 \leq x \leq 2$ and $4 \leq x \leq 5$. Here the net length of the Pareto optimal region is 2 units in variable space. These disjointed regions are due to the two minima of f_{21}. Schaffer used this problem to test VEGA's ability to find these two disjointed nondominated regions. This problem is chosen for NSGA for the same purpose, and also to test its distributive powers in allocating its population members over these two disjointed regions.

Problem F3 : Two-parameter, unimodal and monotonically decreasing objectives:

This problem is used to test NSGA's ability in optimizing multiparameter functions and distributing the population over infinite nondominated region. This problem was solved by Chankong and Haimes (1983) using a goal vector and weights for objectives. The objectives, f_{31} and f_{32}, are :

$$\begin{aligned} \text{Minimize} \quad f_{31} &= (x_1 - 2)^2 + (x_2 - 1)^2 + 2, \quad \text{and} \\ \text{Minimize} \quad f_{32} &= 9x_1 - (x_2 - 1)^2. \end{aligned} \tag{6.3}$$

In this thesis, the first two problems are considered for comparative analysis of VEGA and NSGA. Since these functions are test functions, some problem knowledge is used to fix σ_{share} parameter. In a later simulation, the knowledge is waived and σ_{share} is computed according to equation 5.4.

The objectives f_{31} and f_{32} are plotted in variable space as shown in figure 6.3. The first objective is a smooth unimodal function which has a minimum at $x_{31}^* = \{2, 1\}^{\text{T}}$. The second objective decreases monotonically with decreasing x_1 and increasing $|x_2|$. This can be observed in figure 6.4 (contour map). The contours of first function are concentric with the center at (2,1). This function increases with increasing diameter. The second function (parallel parabolas) constantly decreases along the line $x_2 = 1$ towards decreasing x_1. Careful observation reveals that the *tangential points* of circles

and parabolas dominate all other points. This is because any such tangential point is better in second objective than all other points belonging to the same circle (same f_{31} value). These tangential points are Pareto-optimal points. Therefore, Pareto-optimal points may be found by equating the slopes (first differentials) of contour curves at common points :

$$\left(\frac{dx_2}{dx_1}\right)_{\text{from } f_{31}} = \left(\frac{dx_2}{dx_1}\right)_{\text{from } f_{32}}$$

or,

$$-\frac{(x_1-2)}{(x_2-1)} = \frac{9}{2(x_2-1)}.$$

Assuming $x_2 \neq 1$, and solving yield,

$$x_1 = -2.5.$$

Thus the Pareto-optimal region is represented by the straight line $x_1 = -2.5$. Since the second objective, f_{32}, monotonically decreases the Pareto-optimal region represented by straight line $x_1 = -2.5$ is unbounded. At the singular points, $x_2 = 1$, observation shows that the nondominated points lie between $x_1 = -2.5$ and $x_1 = 2$. Therefore the nondominated region is an infinite straight line, $x_1 + 2.5 = 0$, and segment of line $x_2 = 1$ from $x_1 = -2.5$ to $x_1 = 2$. This region is marked in figure 6.4.

To make the comparisons better, exact nondominated points found above are compared with the simulation results.

6.2 Performance measure

In order to investigate how well NSGA and VEGA have distributed individuals over the nondominated region, chi-square-like deviation form distribution measure developed by Deb (1989) is used. The performance measure, ι, is defined as

$$\text{Performance measure, } \iota = \sqrt{\sum_{i=1}^{q+1}\left(\frac{n_i - \bar{n_i}}{\sigma_i}\right)^2}, \tag{6.4}$$

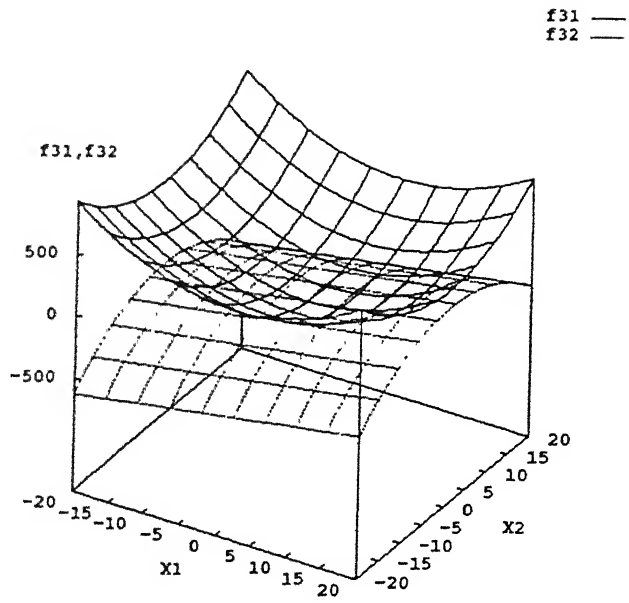


Figure 6.3: Problem F3 drawn in variable space.

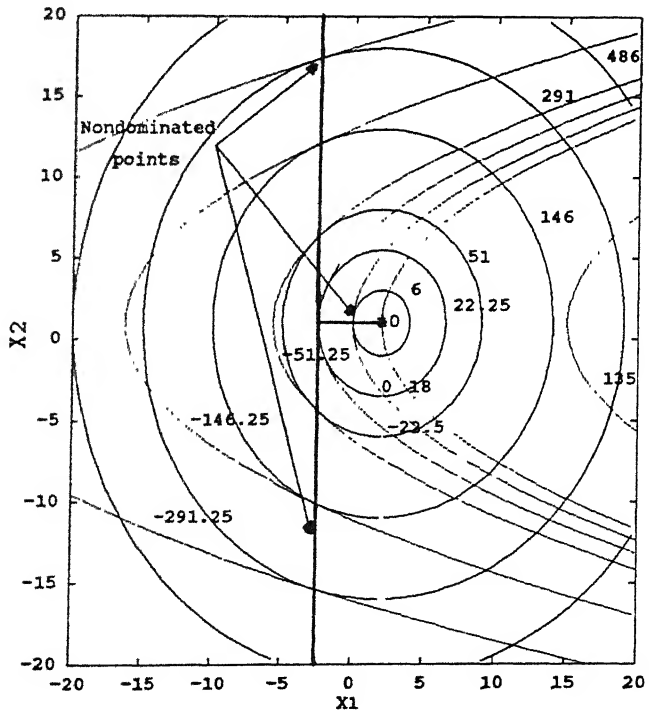


Figure 6.4: Contour map of problem F3.

where q is the number of desired optimal points and $(q+1)$-th subspace is the dominated region. n_i is actual number of individuals serving i-th subspace (niche) of nondominated region. $\bar{n}_i$ is expected number of individuals serving i-th subspace of nondominated region. and σ_i^2 is the variance of individuals serving i-th subspace of nondominated region.

Using probability theory it was estimated elsewhere (Deb 1989) that

$$\sigma_i^2 = n_i\left(1 - \frac{\bar{n}_i}{\mathrm{P}}\right), \qquad i = 1, 2, \ldots, q,$$

where P is the population size. Since it is not desirable to have any individual in the dominated region ($(q+1)$-th subspace), $\bar{n}_{q+1} = 0$. That study also showed that $\sigma_{q+1}^2 = \sum_{i=1}^{q} \sigma_i^2$. If distribution is exactly uniform, performance measure $\iota = 0$. Therefore an algorithm with good distributing capability is characterized by a low deviation measure. This measure can be used to compare the distributive powers of NSGA and VEGA. An ideal distribution can be taken as an uniform spread of population over the nondominated regions.

6.3 Simulation Results

In this section, experimental results of NSGA and VEGA are presented. In all simulations for the first two problems, the GA parameters used in the experiments are as follows:

Maximum generation	:	500
Population size	:	100
String length (binary code)	:	32
Probability of crossover	:	1.0
Probability of mutation	:	0.0

The parameters are held constant across all runs. Unbiased initial population is generated randomly spreading over entire variable space in consideration. To make

the comparison fair, exactly the same initial population has been used in VEGA and NSGA. To confirm and recheck the results, each experiment is repeated five times with different initial population and the average performance is presented in each case.

6.3.1 Problem F1

The population drift in case of VEGA and NSGA, was already compared in figures 5.2 through 4.5 in chapters 4 and 5. In order to study the distribution pattern better, the nondominated search space $(0, 2)$ is divided into 10 equal sub-regions. Figures 6.5 and 6.6 show plots drawn with number of individuals in each sub-region and generation number. In case of VEGA (figure 6.5), after some generations, some of the sub-regions do not have representation at all. These sub-regions represent middling or utopian individuals. Observations based on a number of simulation results reveal that *at most* three sub-regions are populated by VEGA. These are the points around individual optima. Whereas, in case of NSGA (figure 6.6), the number of individuals in each sub-region fluctuated around value 10, which is exactly the expected number of points at each sub-region. It is very important to note that none of the sub-regions have zero number of individuals. To quantify this distribution capability of population over nondominated regions, the chi-square-like performance measure is calculated.

To analyze the distribution using this performance measure, the nondominated region is divided into same 10 equal sub-regions (each having a length 0.2 units in variable space). Since a population of 100 individuals is used, the expected number of points per sub-region $(\bar{n}_i)$ is 10[1] with a variance $\sigma_i^2 = 9$. Therefore, the expected variance of dominated individuals $\sigma_{11}^2 = 90$. The actual number of individuals in each sub-region are counted and deviation measure is calculated using equation 6.4.

[1] Assuming that equal number of points are probable in each subregion, in general for any problem this may not be true.

Figures 6.7 and 6.8 show deviation measure versus generation number for VEGA and NSGA applied on F1. Figure 6.7 shows the average performance of five runs with different initial populations while taking the same initial population for VEGA and NSGA. Initially both methods start with high performance measure because the initial population is spread over entire variable space, with less number of individuals in the nondominated region. VEGA's increasing measure with generation indicates its poor distributing ability. The initial descent is due to the convergence of population towards nondominated region. At the same time NSGA with $\sigma_{\text{share}} = 0.1$ (induced number of niches in nondominated region, $q = 10$), fluctuated at a low deviation measure. This is continued until generation 500 which is long enough to justify the stability of population distribution in 10 sub-regions. This shows the ability of NSGA in distributing population over nondominated region.

In order to investigate how sensitive the NSGA results on σ_{share} values, a number of σ_{share} values are tried. Figure 6.8 shows performance of NSGA with different σ_{share} values. To make a fair comparison, the initial population is taken to be same in all cases. There is not much difference in performance with $\sigma_{\text{share}} = 0.1$ and $\sigma_{\text{share}} = 0.2$. This shows that both the values resulted in successful distribution of population. But with a considerably high sharing parameter, $\sigma_{\text{share}} = 1.0$, NSGA's performance is very poor. Similar observation can be made in case of negligible σ_{share} or without sharing. It is interesting to observe that although these two cases exhibit increasing measure they are not as bad as VEGA's. This is due to the fact that equal reproductive potential (dummy fitness) is maintained for all nondominated individuals, thereby minimizing the bias against middling points.

These results suggest that NSGA is effective in finding multiple Pareto-optimal solutions and is better than VEGA in that respect.

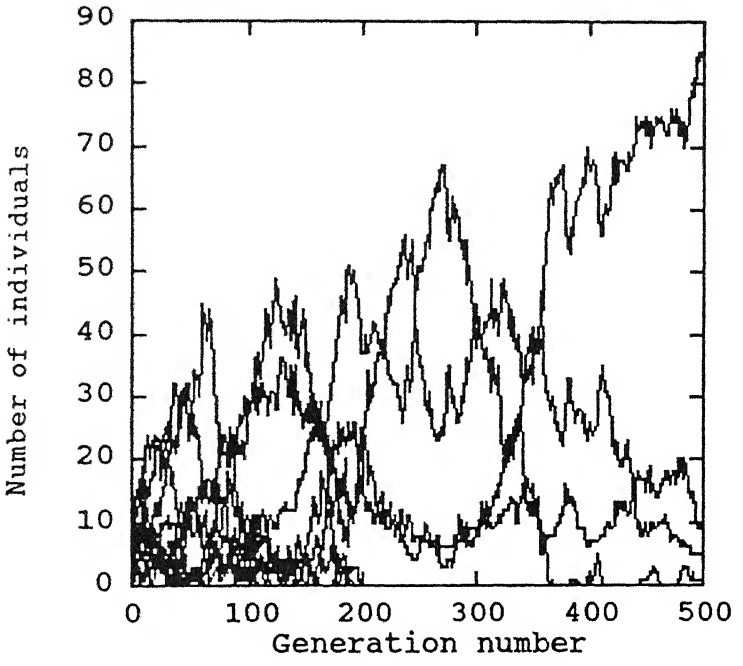


Figure 6.5: Number of individuals in each sub-region versus generation for F1 using VEGA.

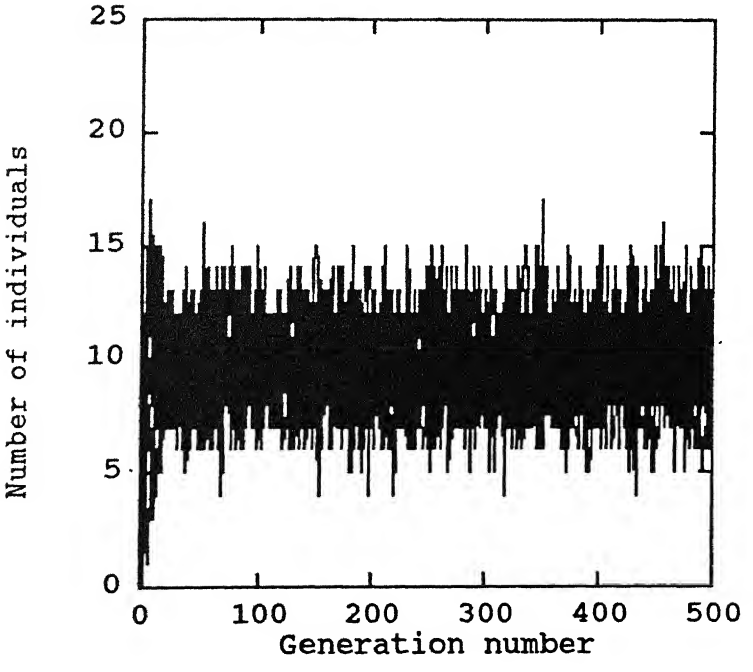


Figure 6.6: Number of individuals in each sub-region versus generation for F1 using NSGA.

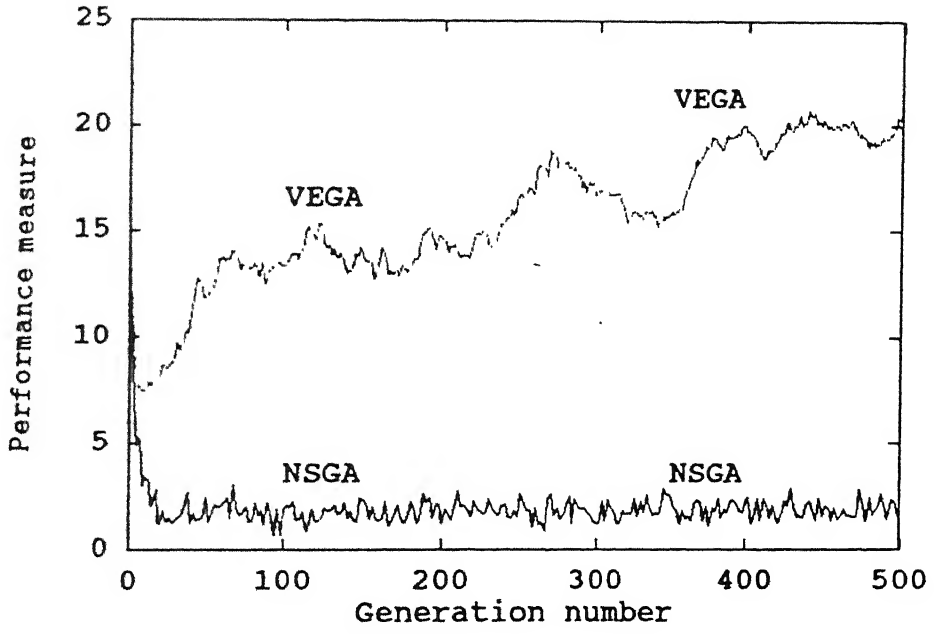


Figure 6.7: Performance measure ι for NSGA and VEGA on problem F1 is plotted versus generation number. An average of five runs is plotted.

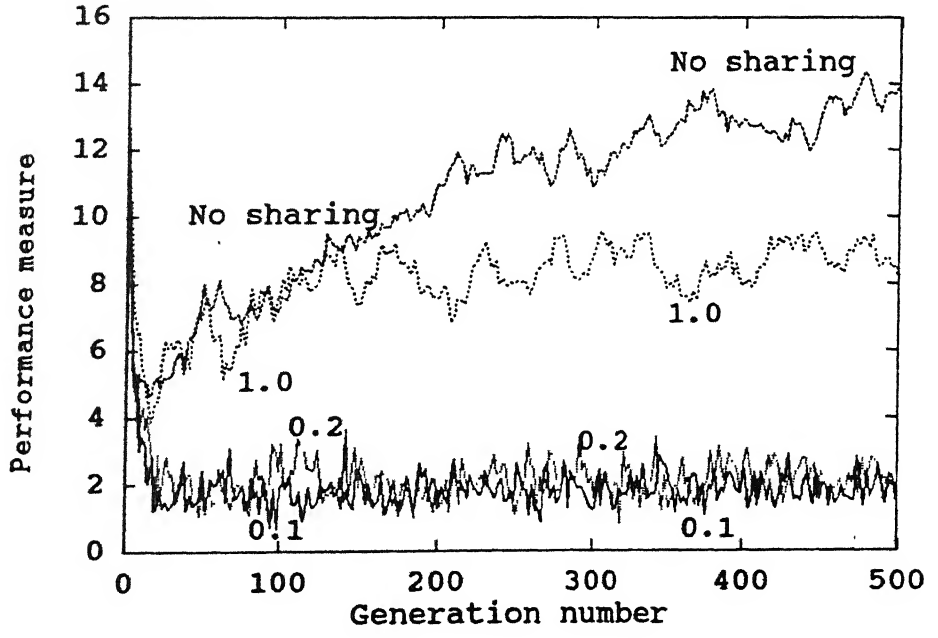


Figure 6.8: Effect of varying σ_{share} values.

6.3.2 Problem F2

The two disjointed Pareto-optima regions are marked in figure 6.2. The initial range for the variable, x, used in the simulations is $[-10, 10]$. The population evolution is shown in figures 6.9 through 6.16. Both algorithms successfully identified disjointed nondominated regions. But the difference in distribution is clearly visible at 100-th and 500-th generations. This result reiterates the ability of NSGA in distributing the population. The nondominated region is divided into 10 sub-regions to analyze distribution of population. Figures 6.17 and 6.18 shows the number of individuals in each sub-region versus generation. VEGA failed to sustain some of the sub-regions in this problem too, whereas NSGA successfully distributed individuals over both disjointed Pareto-optimal fronts. The deviation measure for these algorithms was similar in pattern to that of problem F1. Figure 6.19 compares the performance measure of NSGA and VEGA.

6.3.3 Problem F3

This problem has infinite Pareto-optimal region. This region (except for boundaries) is marked in figure 6.4. This problem was solved by Chankong and Haimes (1983) using a goal programming method. They assumed a goal vector and a weight vector for objectives, and considered the variable space $x_1, x_2 \geq 0$. This completely eliminated the nondominated region $x_1 = -2.5$. The solution is a single point along the line $x_2 = 1$. In order to get other nondominated points, goal vectors as well as weight vectors are to be changed, and the problem has to be formulated again. This clearly shows the dependence of this method on goals and weights. An improper goal may some times result in a non Pareto-optimal solution.

NSGA and VEGA are applied to this problem considering the variable space $-20 \leq x_1, x_2 \leq 20$. Due to this bounded search space the nondominated region

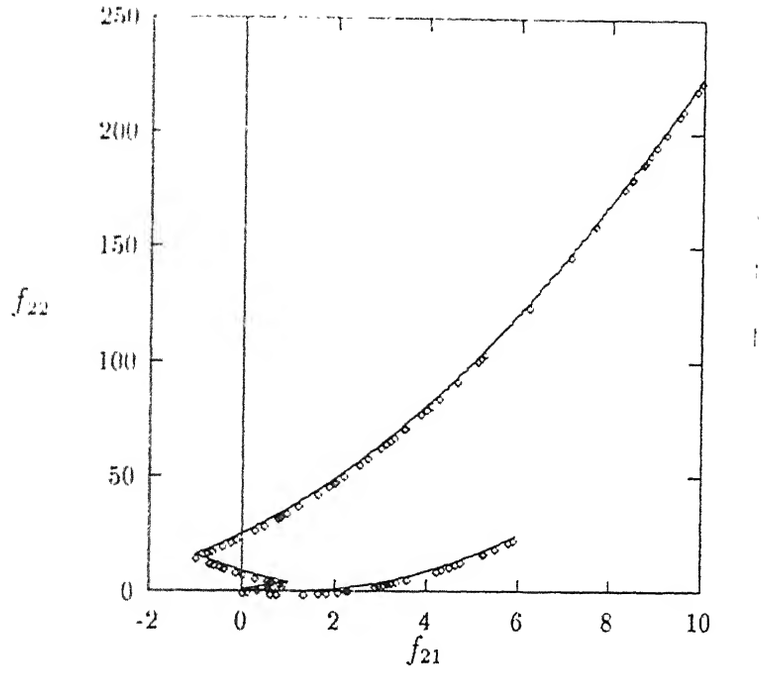


Figure 6.9: Population at generation 0 obtained using NSGA for problem F2.

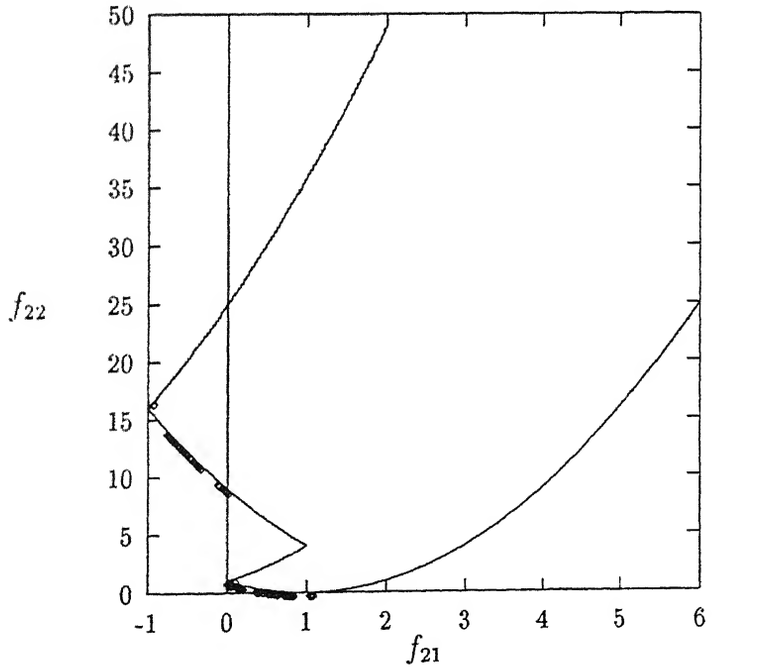


Figure 6.10: Population at generation 10 obtained using NSGA for problem F2.

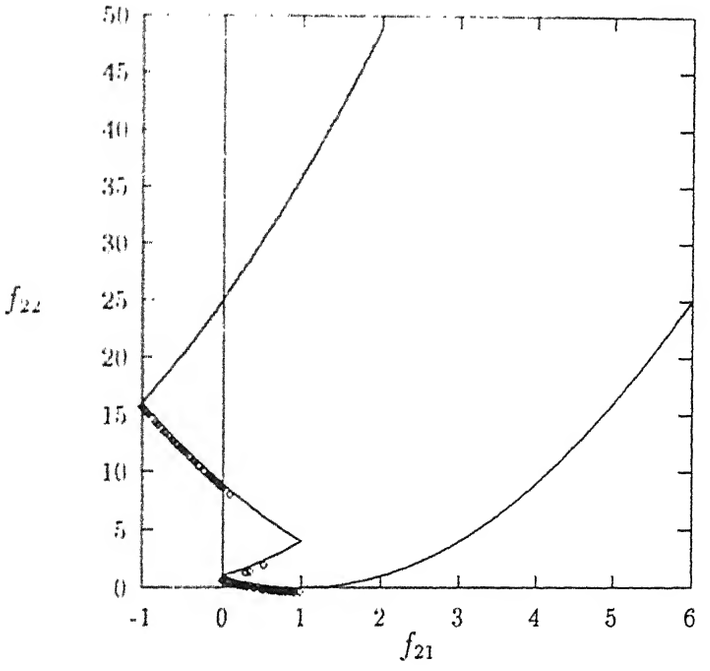


Figure 6.11: Population at generation 100 obtained using NSGA for problem F2.

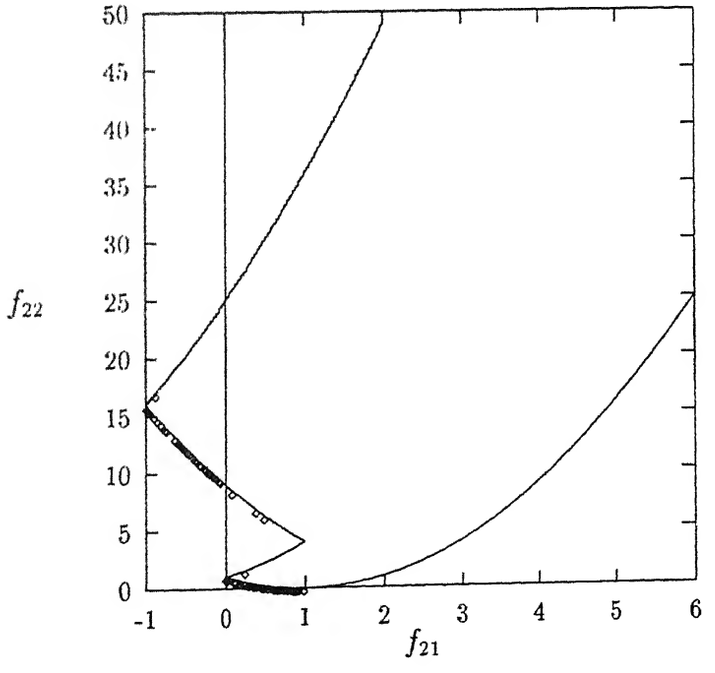


Figure 6.12: Population at generation 500 obtained using NSGA for problem F2.

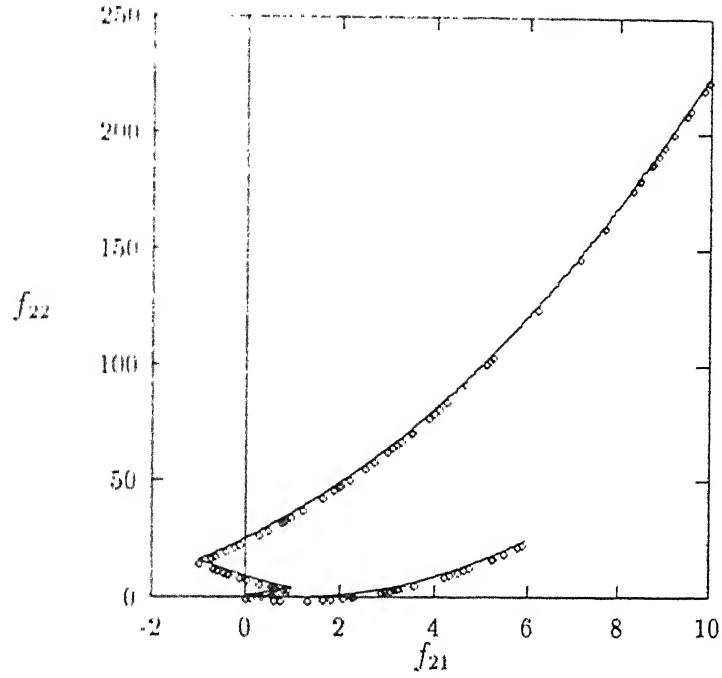


Figure 6.13: Population at generation 0 obtained using VEGA for problem F2.

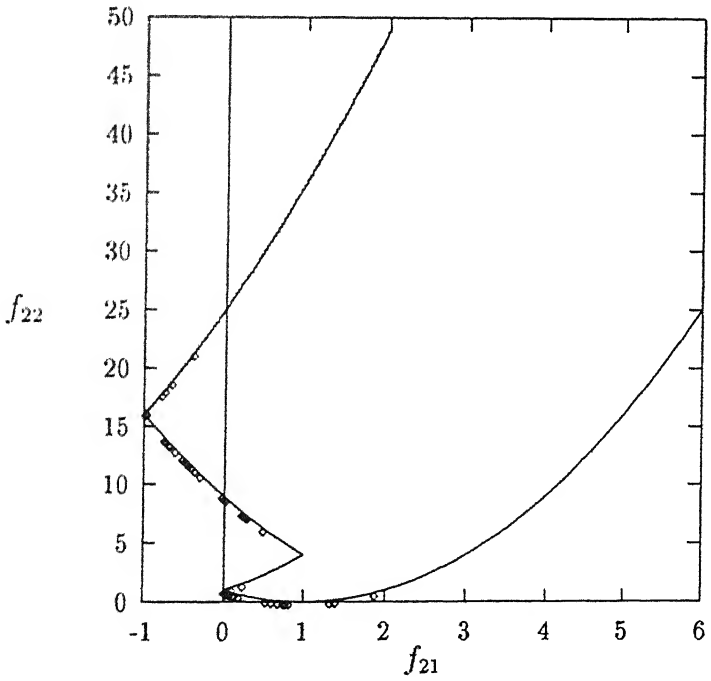


Figure 6.14: Population at generation 10 obtained using VEGA for problem F2.

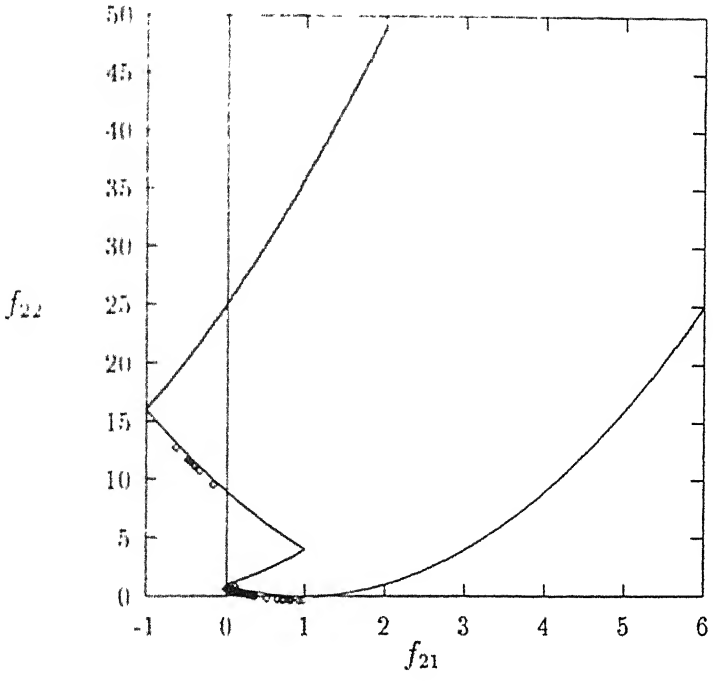


Figure 6.15: Population at generation 100 obtained using VEGA for problem F2.

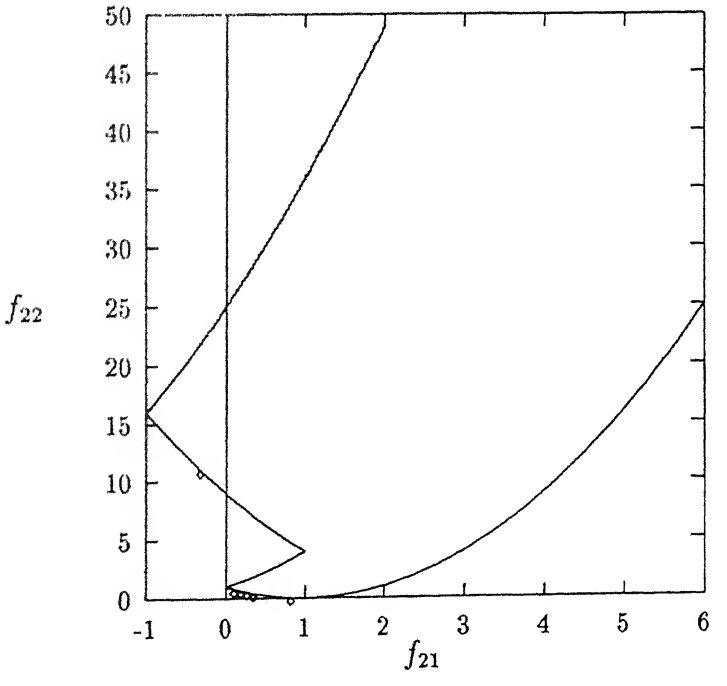


Figure 6.16: Population at generation 500 obtained using VEGA for problem F2.

will be

$$\begin{aligned}
x_1 &= -2.5 \text{ from } x_2 = -20 \text{ to } x_2 = 20\,, \\
x_2 &= 1 \text{ from } x_1 = -2.5 \text{ to } x_1 = 2\,, \\
x_2 &= 20 \text{ from } x_1 = -20 \text{ to } x_1 = -2.5\,, \\
x_2 &= -20 \text{ from } x_1 = -20 \text{ to } x_1 = -2.5\,.
\end{aligned}$$

In all simulations, for this problem, GA parameters used in experiments are :

Maximum generation	:	500
Population size	:	100
String length (binary code)	:	30 (two 15-bit strings for each variable)
Probability of crossover	:	1.0
Probability of mutation	:	0.0

The σ_{share} parameter for NSGA, calculated using equation 5.4, is found to be 8.9 (for inducing 10 niches in the variable space). The experiments are conducted with different σ_{share} values (8.0, 8.5, and 9.0) and with different initial populations. The population movement, in case of NSGA with $\sigma_{\text{share}} = 8.0$, is shown in figures 6.20 through 6.23. For the same initial population the drift in VEGA is shown in figures 6.24 through 6.27. Simulations with the other σ_{share} values are very similar to these results.

The final population (at generation 500) of NSGA is finely distributed along the line $x_1 = -2.5$. At the same time VEGA's population converged towards some subdomains which are nearer to the individual optima. It can be observed that NSGA failed to maintain some nondominated individuals which are on the line $x_2 = 1$. At generation 10, some individuals are present in the population. Their loss in subsequent generations can be attributed to the large σ_{share} value. Since a niche is assumed as a hypersphere (circle in this case) of radius σ_{share}, most of these points fall within a circle (induced peak). Thus, NSGA distributes strings to different niches, which may result in the loss of many closely packed points. This effect can be observed in case of points of the segment $x_2 = 1$. The length of this segment is very small, 4.5

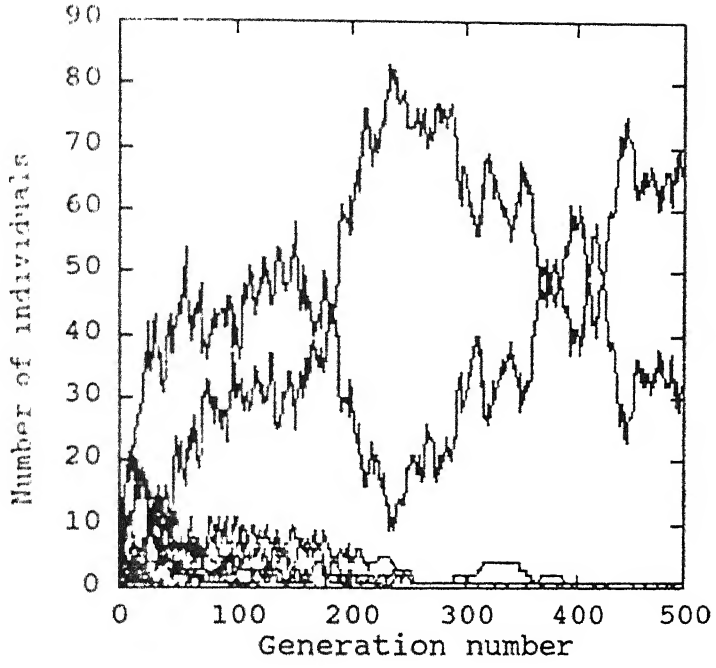


Figure 6.17: Number of individuals in each sub-region versus generation for F2 using VEGA.

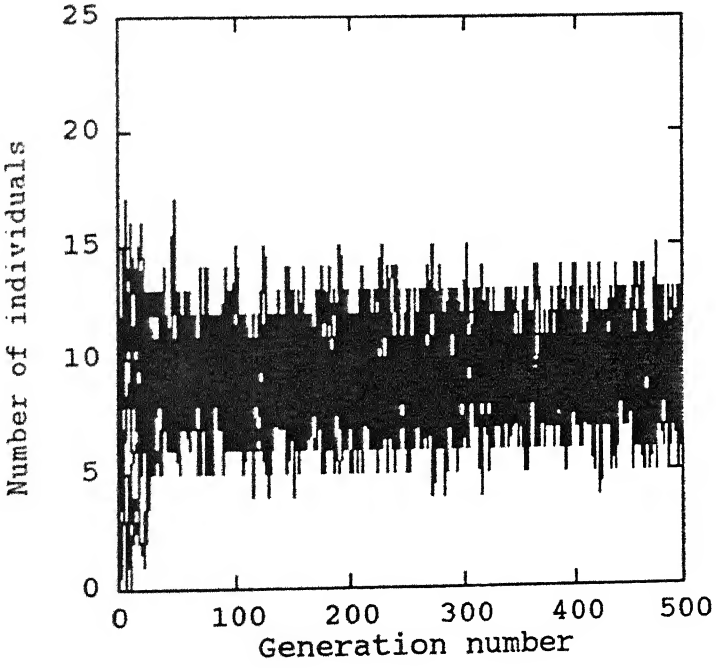


Figure 6.18: Number of individuals in each sub-region versus generation for F2 using NSGA.

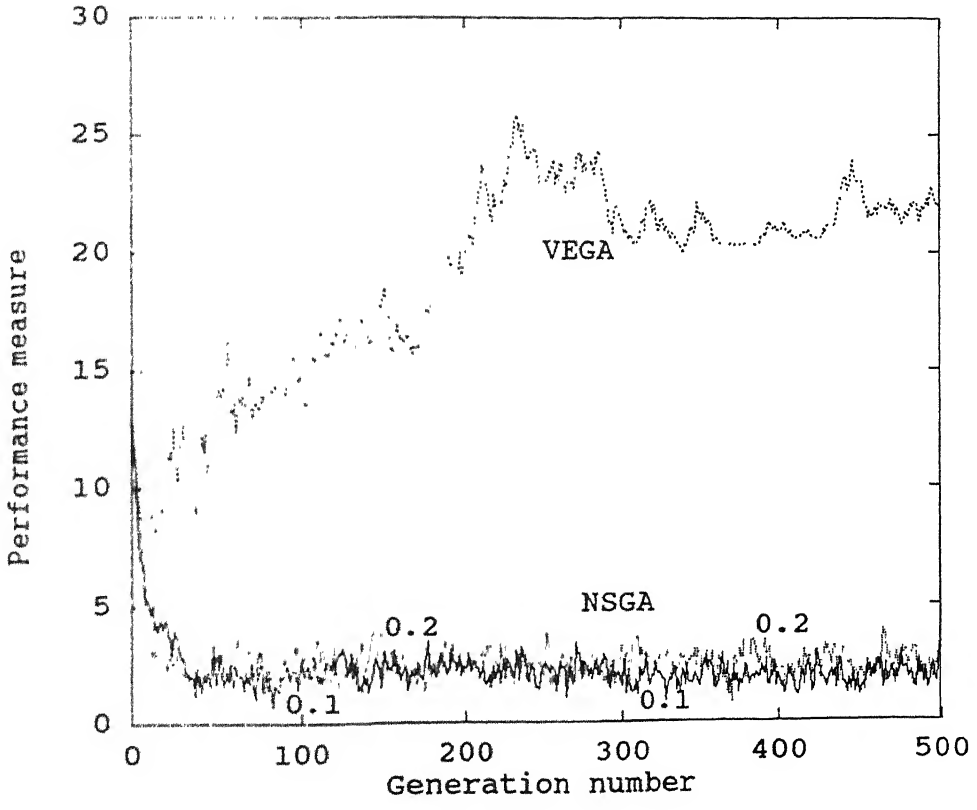


Figure 6.19: Performance measure, ι, for NSGA and VEGA on problem F2.

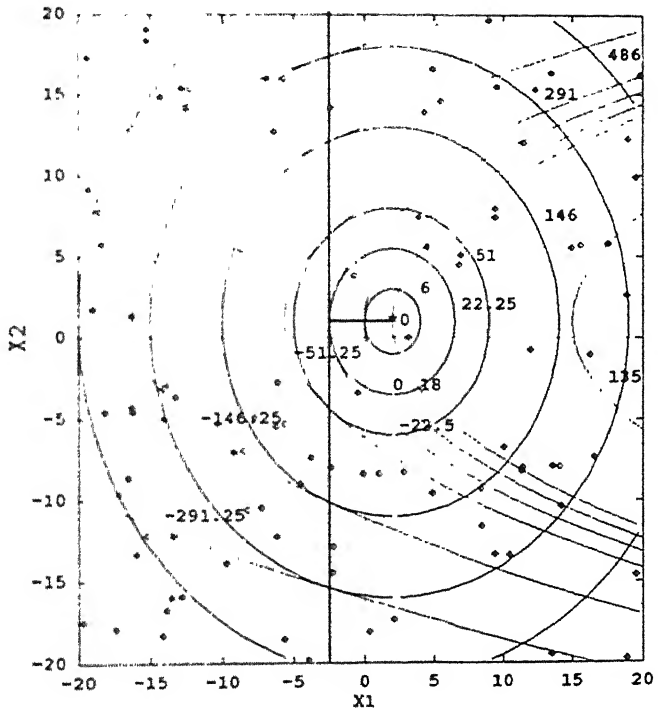


Figure 6.20: Population at generation 0 obtained using NSGA for problem F3.

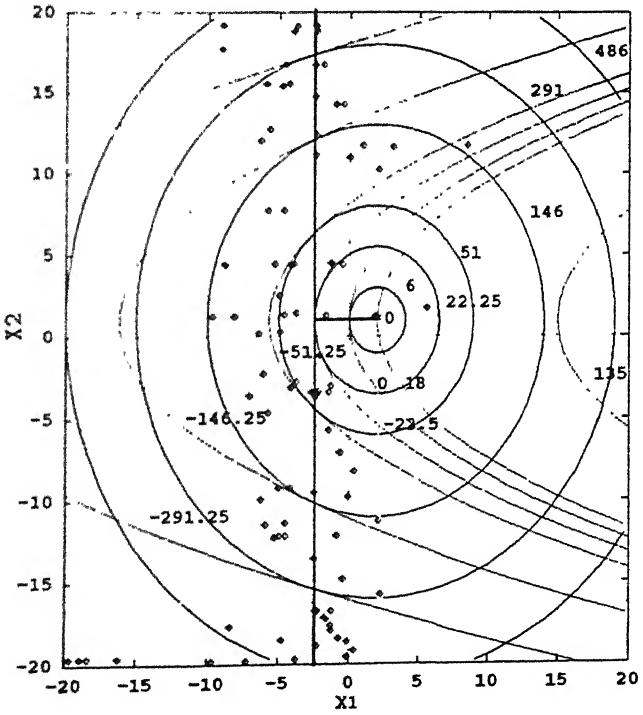


Figure 6.21: Population at generation 10 obtained using NSGA for problem F3.

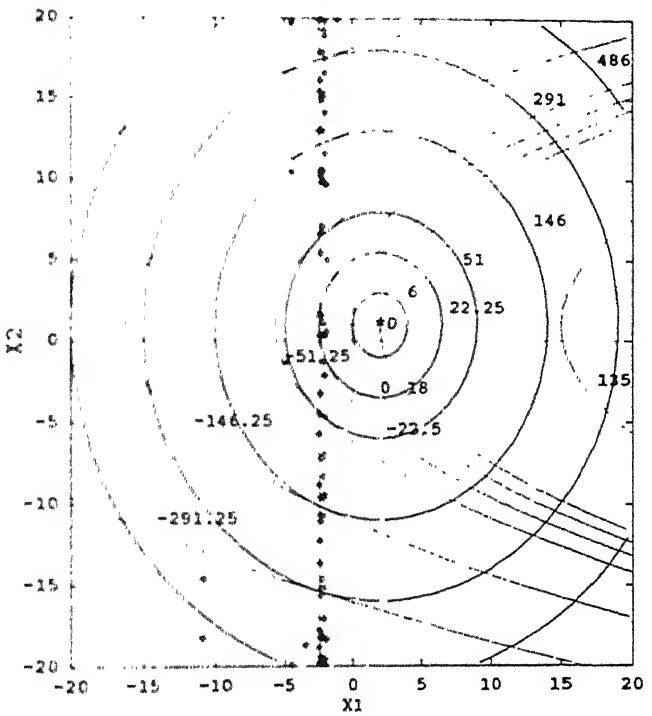


Figure 6.22: Population at generation 100 obtained using NSGA for problem F3.

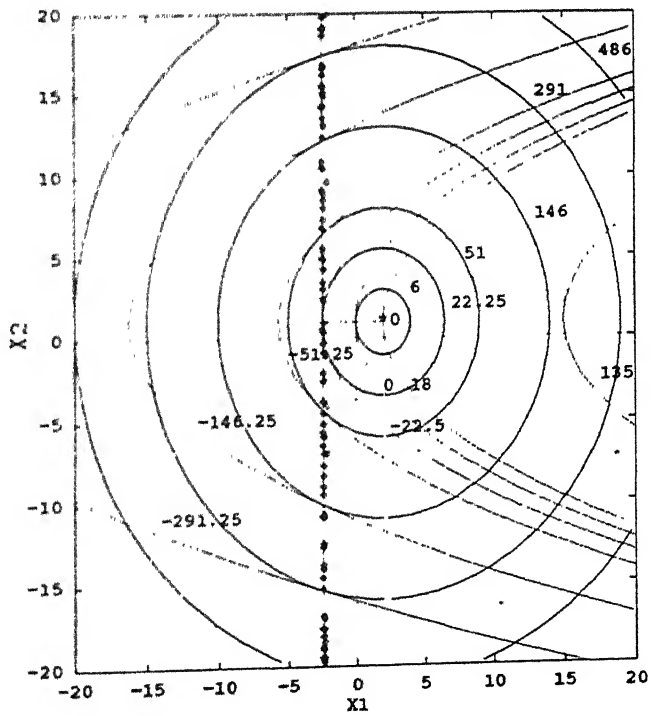


Figure 6.23: Population at generation 500 obtained using NSGA for problem F3.

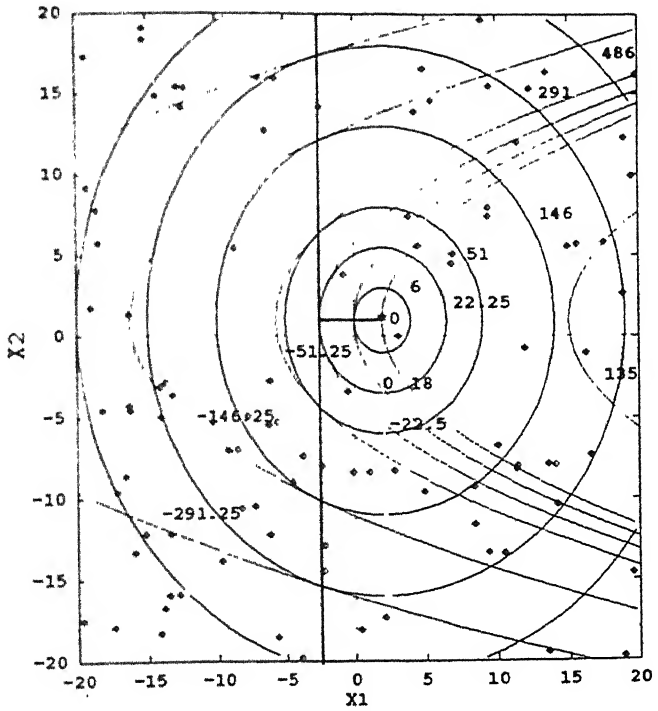


Figure 6.24: Population at generation 0 obtained using VEGA for problem F3.

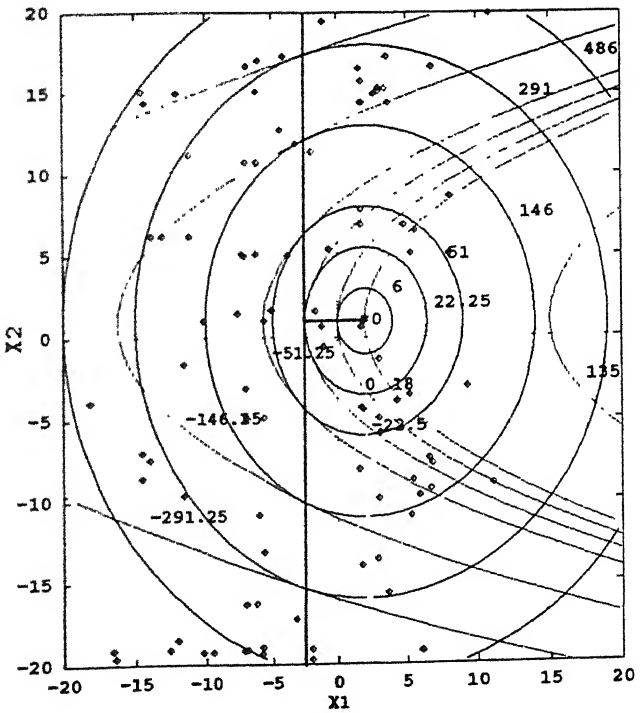


Figure 6.25: Population at generation 10 obtained using VEGA for problem F3.

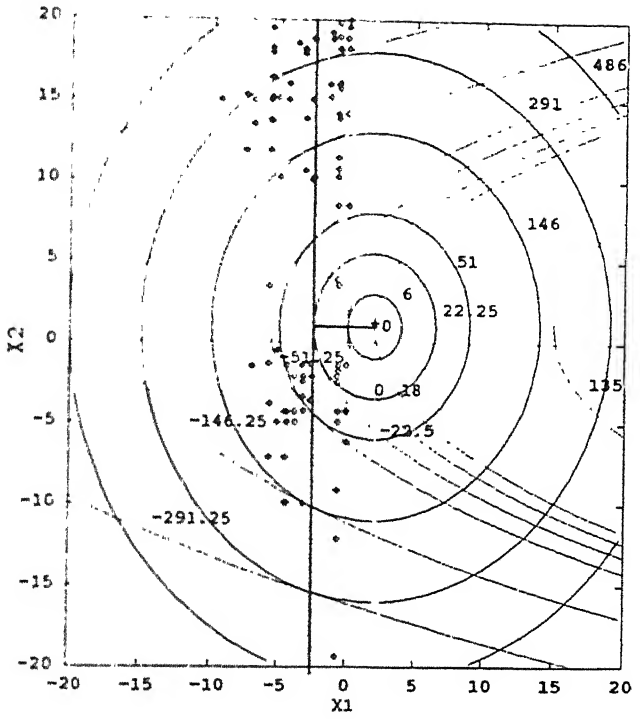


Figure 6.26: Population at generation 100 obtained using VEGA for problem F3.

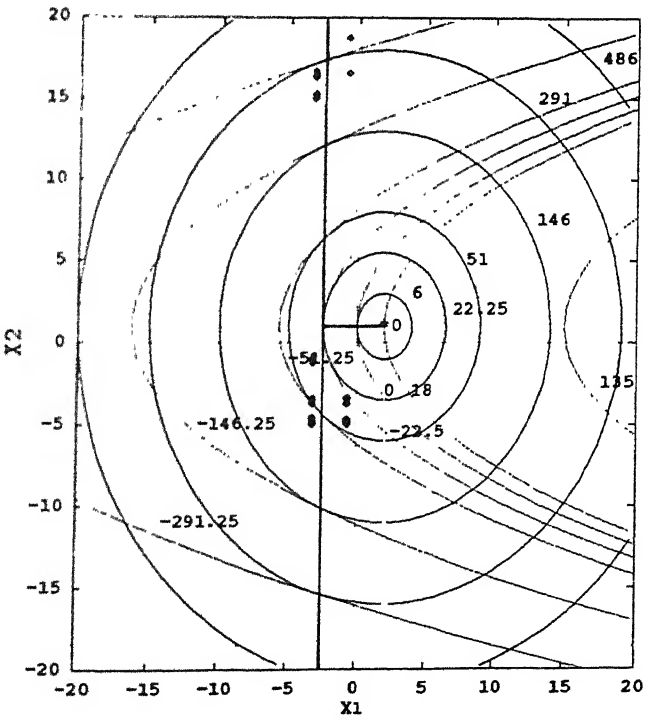


Figure 6.27: Population at generation 500 obtained using VEGA for problem F3.

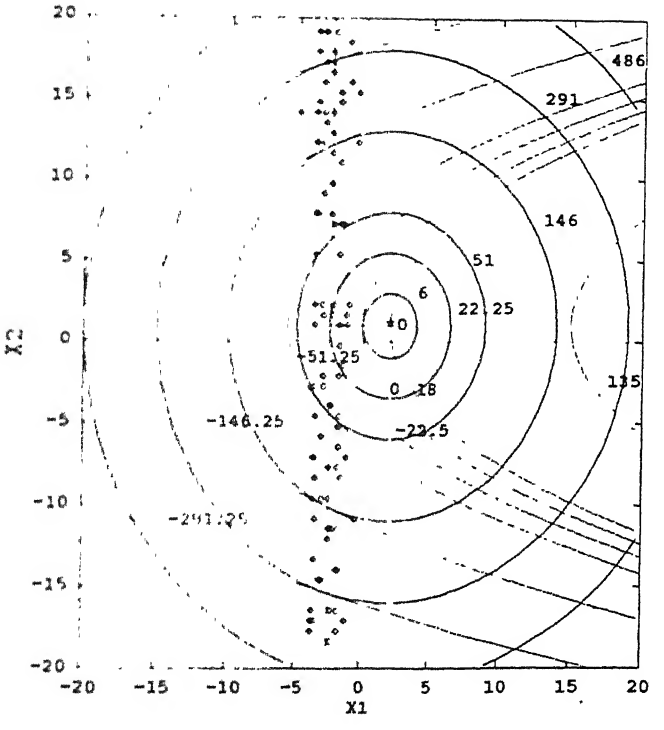


Figure 6.28: Population at generation 700 obtained using NSGA with $\sigma_{\text{share}} = 1$.

Chapter 7

Multiobjective Truss Optimization

In this chapter, NSGA is applied on two multiobjective truss optimization problems. These two problems were solved by Koski (1988) using traditional methods. NSGA's results are compared with these solutions.

7.1 Introduction

Trusses are load carrying structures which appear frequently in a variety of industrial applications. An engineer always tries to save the material by minimizing weight (or volume) of a structure. Weight reduction of a structure plays a major role in minimizing overall installation cost of a factory. Often, the spans of these structures are wide and thus unfavorable deformations may occur. Some times these deflections may exceed specified limits. In addition, many dynamic and crack propagation effects can be avoided indirectly by preventing large displacements. Usually this problem is handled by formulating a single optimization problem — minimization of volume of the structure subjected to displacement constraints. But such constraints require the knowledge of limits for the displacements beforehand. Often the solutions may fall along constraint boundaries. Such solutions are, sometimes, undesirable. These difficulties suggest treating the most critical displacements as design objectives. This

results in a multiobjective optimization problem, with the following objectives :

$$\text{Minimize } \mathbf{f(x)} = \{ V, \Delta_1, \Delta_2, \cdots, \Delta_m \}^T,$$

where x is the design vector, V is the material volume of truss, and $\Delta_i, i = 1, 2, \ldots, m$, is the i-th nodal displacement. The material volume, V, of a n-bar truss, can be expressed as

$$V = \sum_{i=1}^{n} L_i A_i ,$$

where L_i is the length of i-th element and A_i is the area of cross section of that element. The cross section of an element of a truss, is assumed to be uniform over its length. The design variables of a truss can be length, cross sectional area, and some characteristic variables, which may include topological changes of structure, the angles between elements, etc. In this chapter two multiobjective truss problems, one with cross sectional areas as design variables and the other with areas and topological design variables, are considered.

The solutions to this optimization problem will be the compromise alternatives that minimize both volume and critical displacements. A designer wants as many different designs as possible to select the best solution that suits his problem. To present such a set of solutions, NSGA may become a successful tool. The truss design is subjected to its feasibility (in case of changing topology) and the stresses induced in its elements. The induced stresses should not exceed the allowable limiting values. These stresses and the nodal displacements are calculated using finite element methods. This approach is briefly discussed in the following section.

7.2 Analysis of Trusses Using Finite Elements

A truss is assumed as a number of bar-elements connected by hinges to each other and by supports to the base. An element of a truss is considered as a one-dimensional axial bar in the local coordinate system, as shown in figure 7.1 (Chandrupatla and

Belegundu, 1991). The local coordinate system consists an axis that coincides with the elemental axis.

The local stiffness matrix, $\mathbf{k}_e$, is given by

$$k_e = \frac{E_e A_e}{l_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$$

where E_e is the Youngs modulus of element e, A_e is the cross sectional area and l_e is the element length. Assuming the element makes a positive angle, θ, with the global axis, the global stiffness matrix can be found (Bathe, 1990) :

$$K_e = \frac{E_e A_e}{l_e} \begin{bmatrix} \cos^2\theta & \cos\theta\sin\theta & -\cos^2\theta & -\cos\theta\sin\theta \\ & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ & \text{symmetric} & \cos^2\theta & \cos\theta\sin\theta \\ & & & \sin^2\theta \end{bmatrix}$$

The truss is analyzed by assembling all elemental equations and by applying boundary conditions, before solving the matrices for unknown displacements. The assembled matrix equation can be written as

$$\mathbf{K}\,\mathbf{u} = \mathbf{F},$$

where $\mathbf{K}$ is global stiffness matrix (assembled), $\mathbf{u}$ is displacement vector, and $\mathbf{F}$ is the load vector. The unknowns, $\mathbf{u}$, are solved from the above equations, using a numerical method, after satisfying the boundary conditions. The force in each member is calculated from individual elemental equations, after solving the nodal displacement vector. These forces are used to compute the stresses induced in each element. A FORTRAN code has been developed to implement this approach to analyze a truss.

7.3 Four-Bar Truss Problem

A Four-bar truss problem with cross sectional area as design variables, and objectives as volume and a critical nodal displacement, is considered as an application for NSGA.

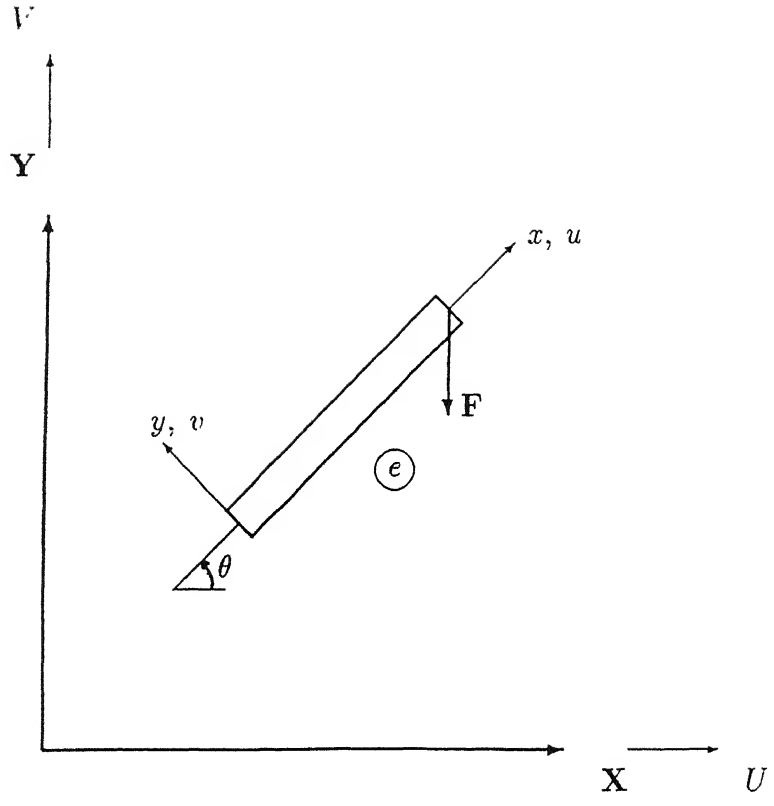


Figure 7.1: Truss element shown in local and global coordinates. U and V are nodal displacements in global coordinate system XY, and u and v are nodal displacements in local coordinate system xy.

The structure and the loading, and the displacement criterion are shown in figure 7.2. The figure also shows the dimensions of each element.

The constraints imposed on the two objectives are the stress limits and area bounds. The material Young's modulus is assumed to be $E = 2 \times 10^4$ kN/cm^2. The allowable stress limits are taken as 10 kN/cm^2 in tension and -10 kN/cm^2 in compression. The design space (variable space) considered for each cross sectional area is $0 \leq A_i \leq 5\text{cm}^2, i = 1, 2, 3, 4$. Thus, the two-objective four-bar truss optimization problem formulated as

$$\text{Minimize} \quad V \quad \text{and} \quad \Delta_2 ,$$

subject to

$$\begin{aligned} -10\,\text{kN/cm}^2 \;\leq\; \sigma_i \;\leq\; 10\,\text{kN/cm}^2, \quad & i = 1, 2, 3.4 \\ 0 \;\leq\; A_i \;\leq\; 5\,\text{cm}^2. \quad & i = 1, 2, 3.4 \end{aligned} \tag{7.1}$$

This problem was solved by J. Koski (1988) using a weighted goal programming method, imposing a goal on deflection. He assumed a desirable displacement $\Delta_2 = 0.6$. Table 7.1 shows these results.

These results are obtained by changing the weights for objectives, while imposing same goal on deflection, and repeating the algorithm to get different Pareto-optimal points. It is interesting to note that, fourth and fifth designs are dominated by second design. This shows that, even if different weights are used, Pareto-optimal points may not be found. Another traditional approach to solve multiobjective truss optimization problems can be found in Rao et al (1992). This approach uses a fuzzy goal programming method.

An exhaustive nondominated sorting search has been made, by discretizing the search space, to identify the Pareto-optimal regions. Later, NSGA is applied on this problem with the following parameters.

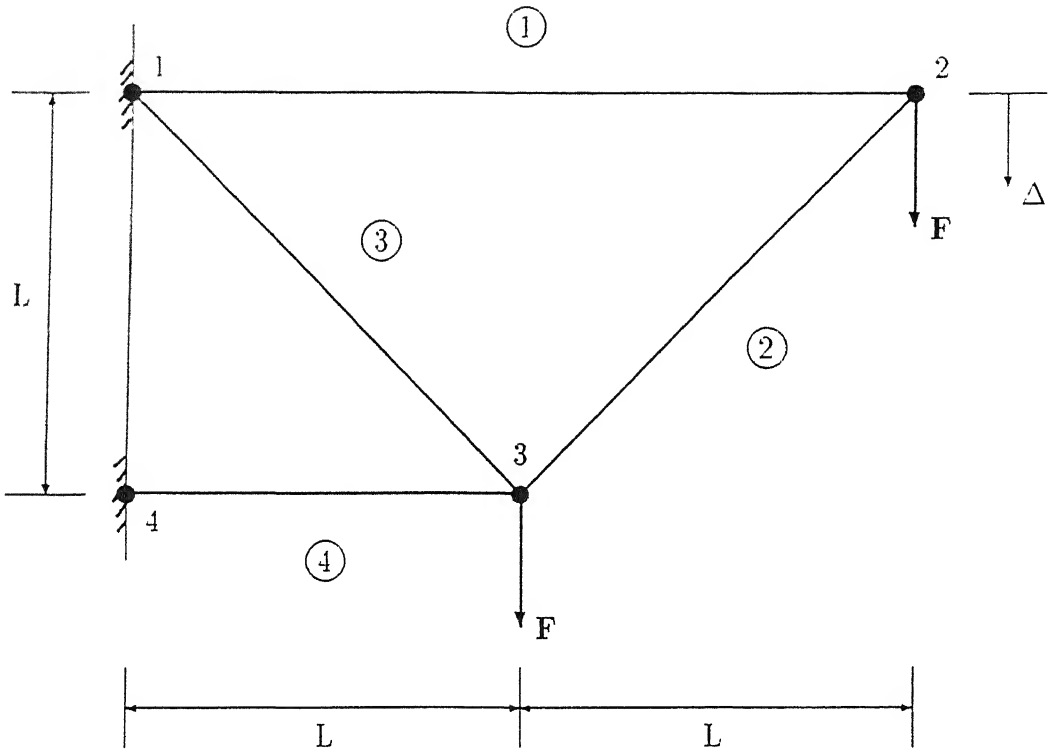


Figure 7.2: Four-bar truss multiobjective optimization problem. The load value F = 10 kN, and L = 200cm.

Table 7.1: Results obtained in J. Koski (1988) using a weighted goal programming method for four-bar truss problem.

Design index	Volume V cm^3	Deflection Δ_2 cm
1	2819	0.617
2	2920	0.596
3	2940	0.592
4	2931	0.602
5	2933	0.600

Maximum generation	:	500
Population size	:	100
String length (binary code)	:	32 (Four 8-bit strings for each variable)
Probability of crossover	:	1.0
Probability of mutation	:	0.0
σ_{share}	:	1.7 (Calculated using equation 5.4)

The σ_{share} is calculated by identifying the feasible region and inducing 10 niches in it. The results are shown in figures 7.3 through 7.6 drawn in performance space. The initial population, figure 7.3, has very few feasible points. Subsequent figures show the Pareto-optimal points are distributed along the first front. The ideal distribution in this case cannot be found because of the complexity of objectives (deflection

function). Experiments revealed that the initial feasible population effects the distribution of population in succeeding generations. Therefore, NSGA is tried with a small mutation (probability of mutation $p_m = 0.01$) and with a crossover probability $p_c = 0.8$, while keeping the other parameters same as above. The final result, population at generation 500, is shown in figure 7.7. In this case the spread of population is more than that in NSGA without mutation. Also, NSGA found some Pareto-optimal solutions found by Koski (table 7.1) using traditional method. These results show the successful implementation of NSGA in above truss problem.

7.4 Eight-Bar Truss Optimization with Changing Topology

Some structures like towers, with a large number of elements, can be able to with stand and serve the purpose even if some panels or elements are removed. This minimization problem is an interesting aspect in engineering design. In topology optimization, the number of members and joints defining the truss itself varies.

An eight-bar truss shown in figure 7.8, is considered. The truss volume and its outer node deflection are the objectives. The design variables are the cross sectional areas and topological variables. In this problem only the number of elements is considered as topological variable. This problem is much more complicated than the previous problem considered, because of more design variables and topological alternatives involved. Early GA application to topology optimization can be found in Krishnamoorthy and Rajeev (1993). They implemented a new approach, called variable length genetic algorithm, VGA. In this method the number of design variable (areas) itself is an unknown. This was implemented by using a control variable that determines the number of panels or an individual's string length. The population consists of strings of different length. Some single-objective topology optimization problems were solved by them, using this approach.

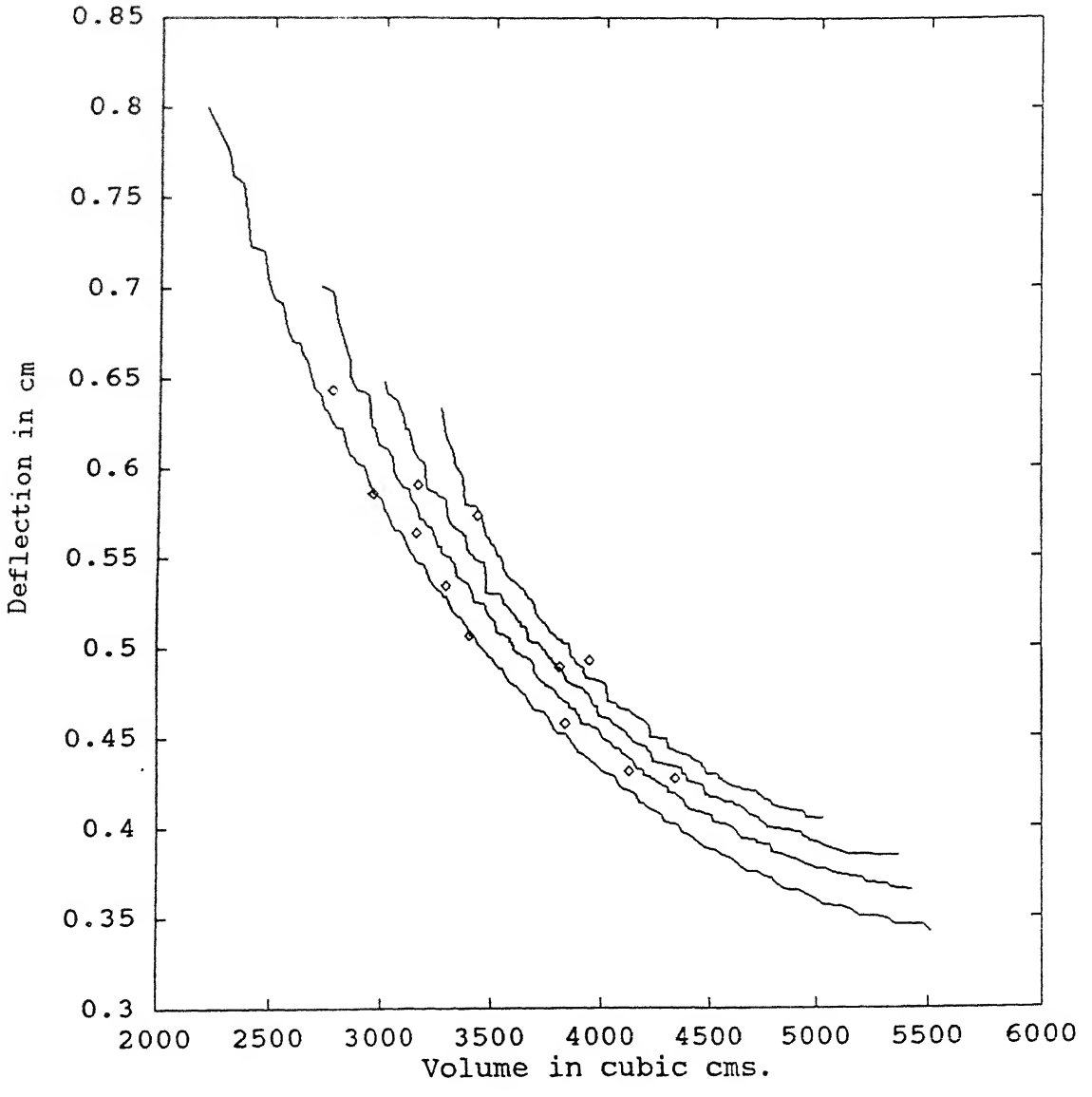


Figure 7.3: Population at generation 0 obtained using NSGA for four-bar truss problem.

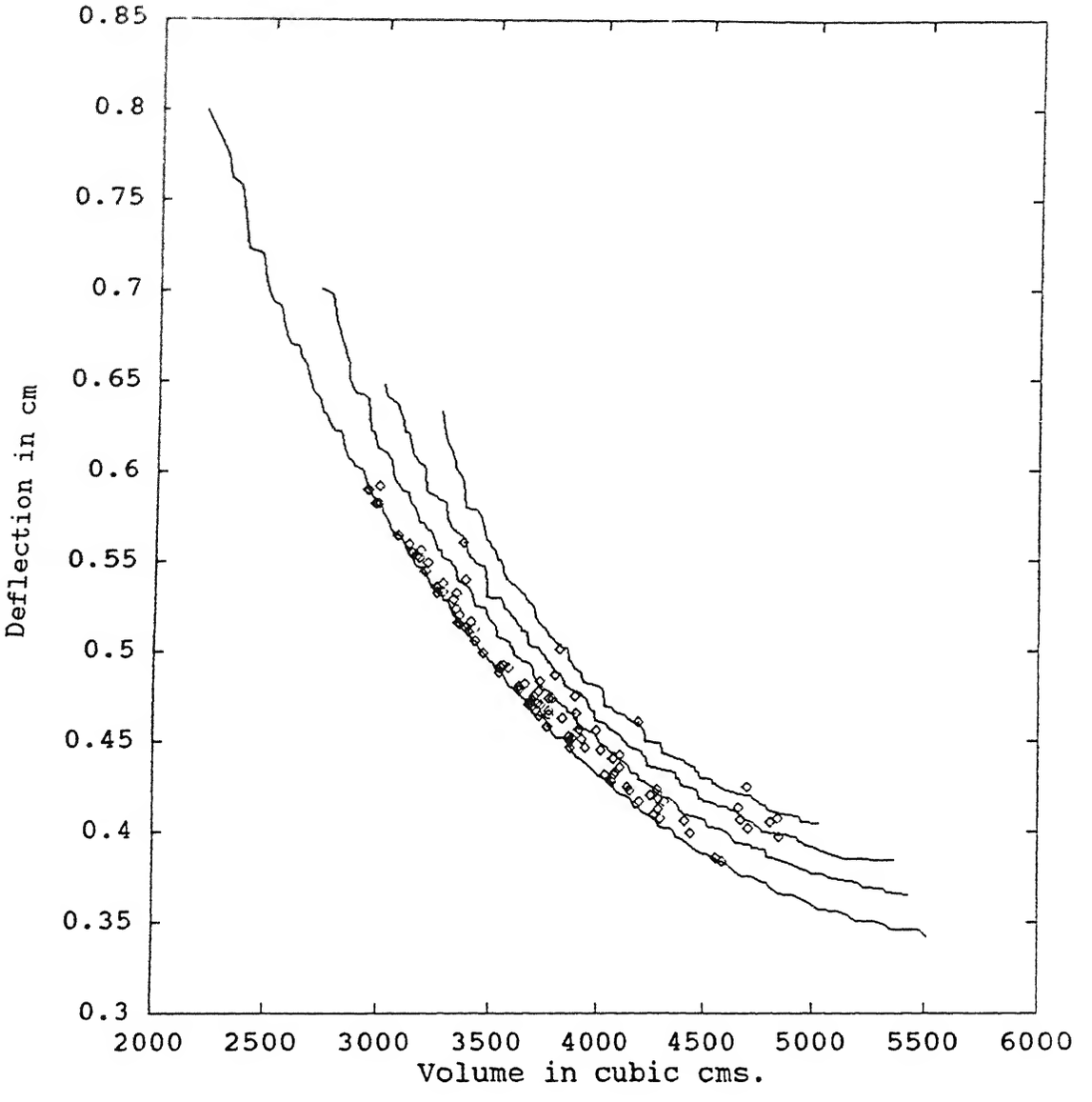


Figure 7.4: Population at generation 10 obtained using NSGA for four-bar truss problem.

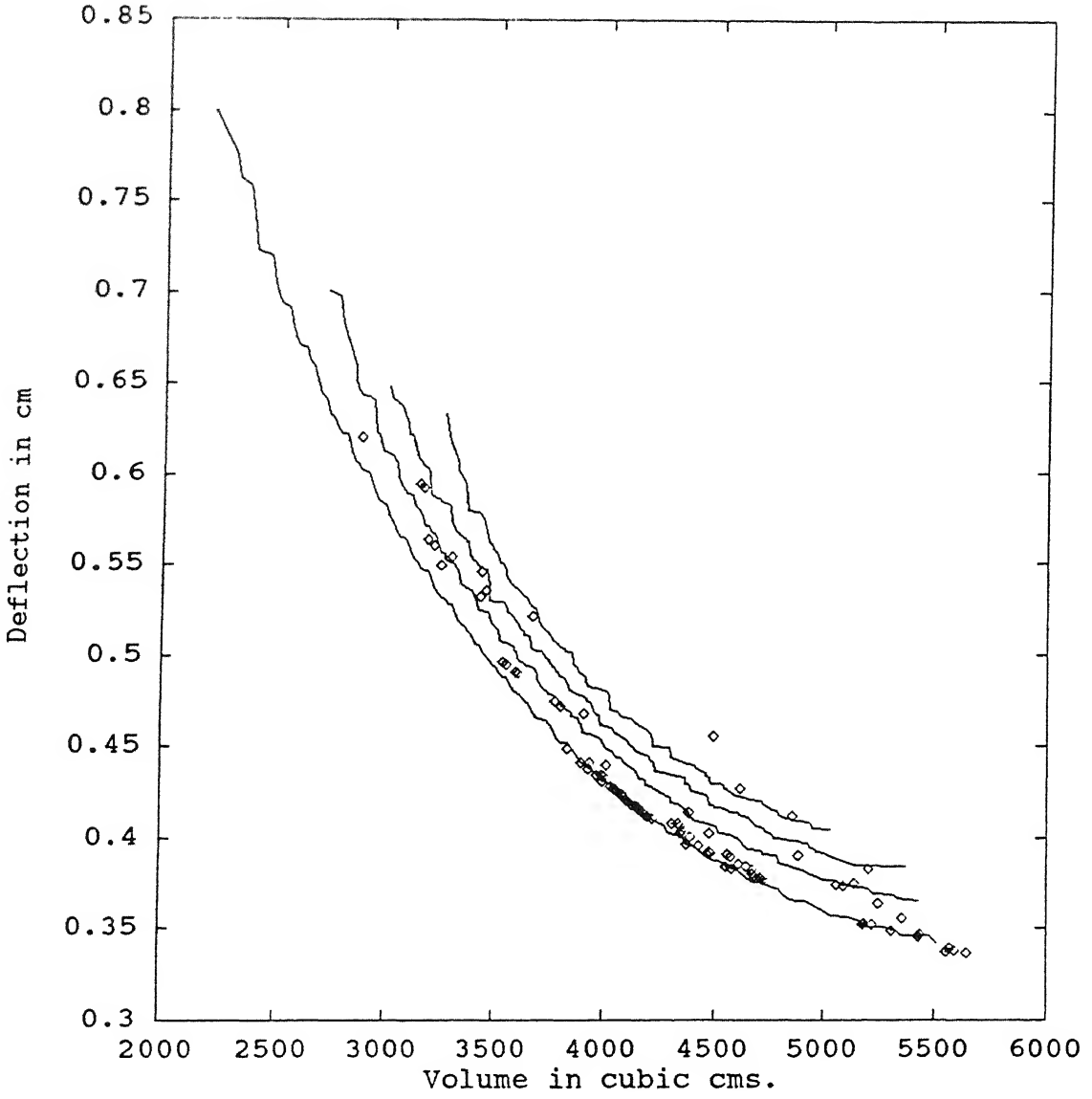


Figure 7.5: Population at generation 100 obtained using NSGA for four-bar truss problem.

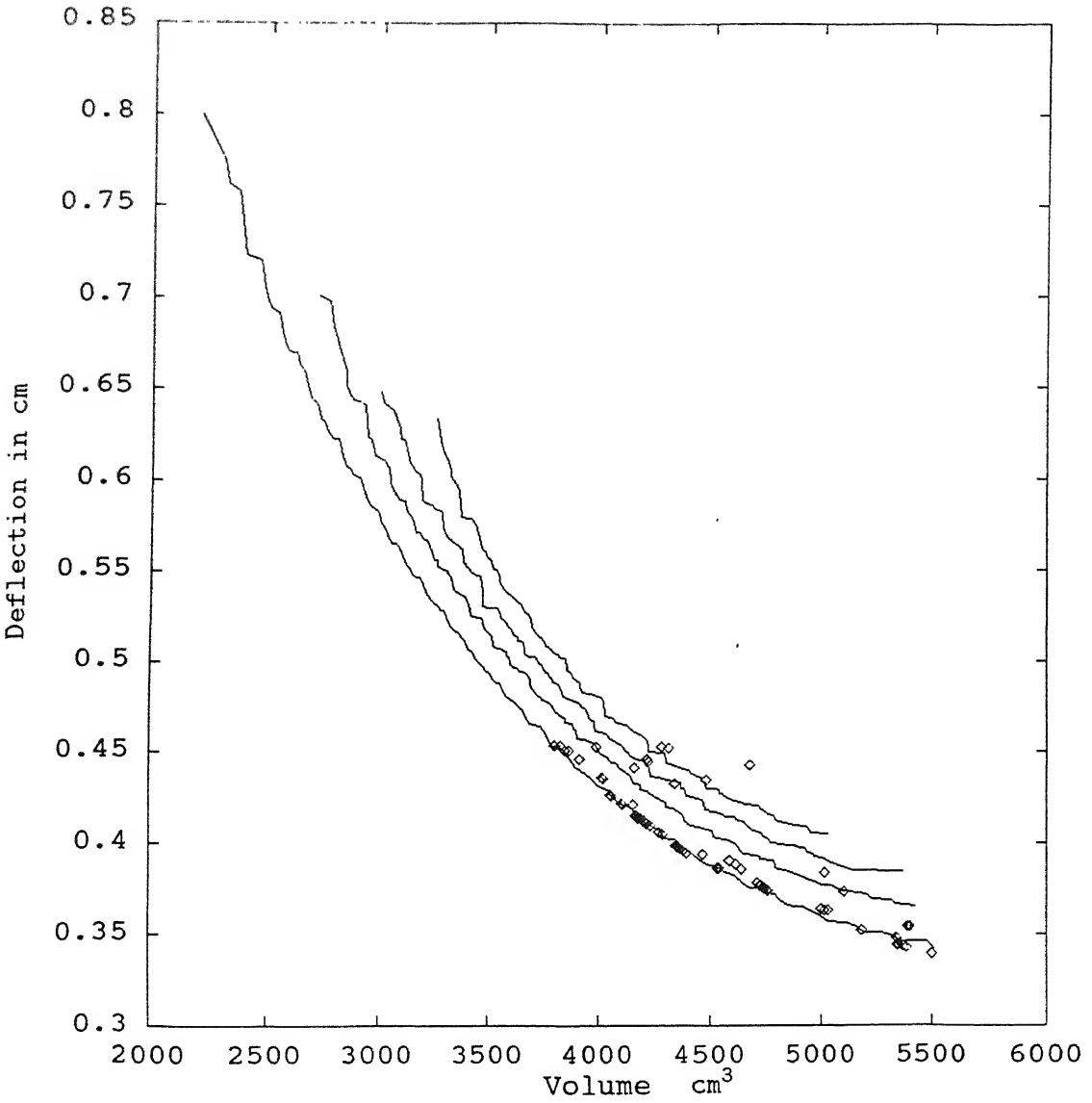


Figure 7.6: Population at generation 500 obtained using NSGA for four-bar truss problem.

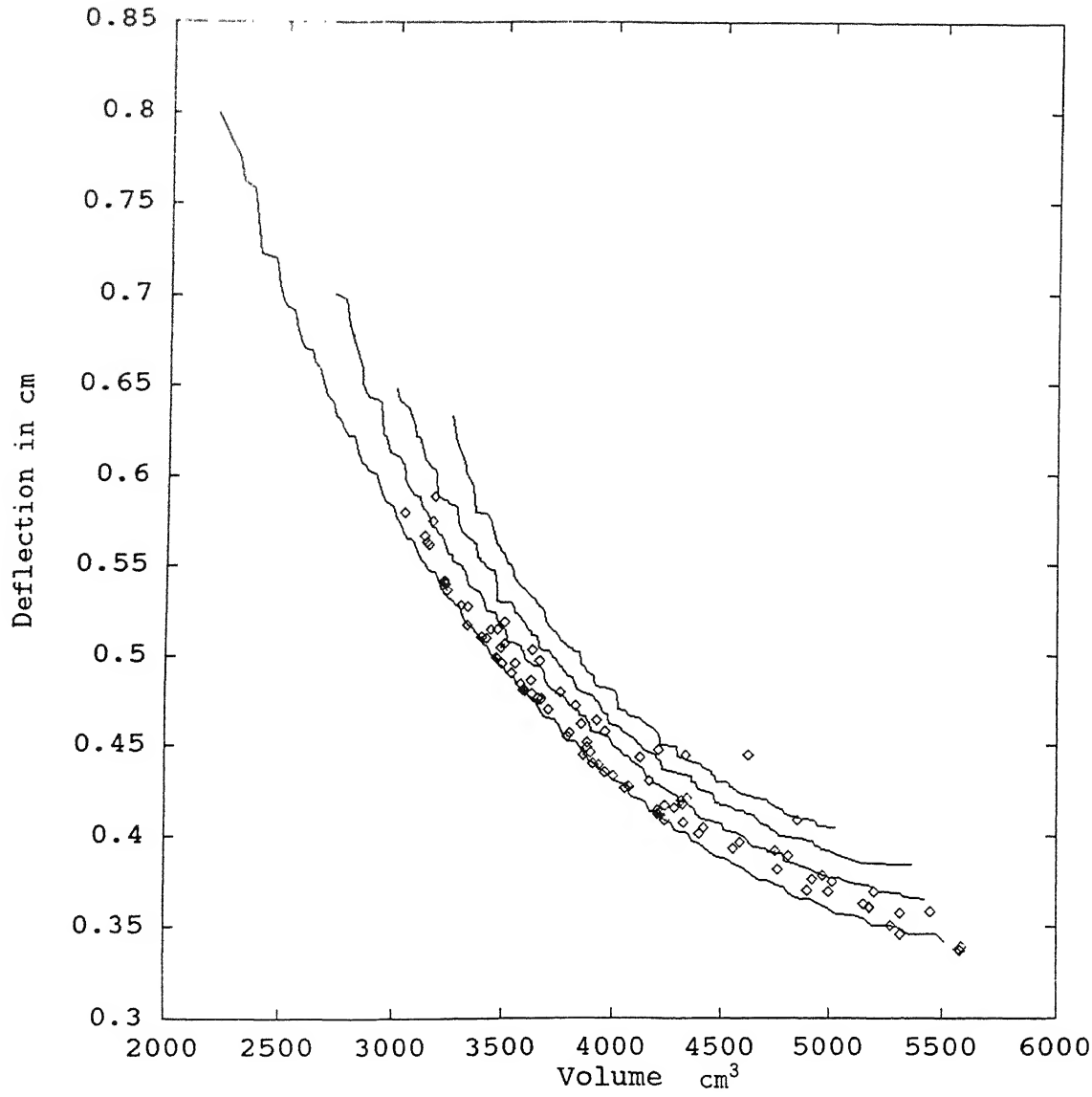


Figure 7.7: Population at generation 500 obtained using NSGA with mutation for four-bar truss problem.

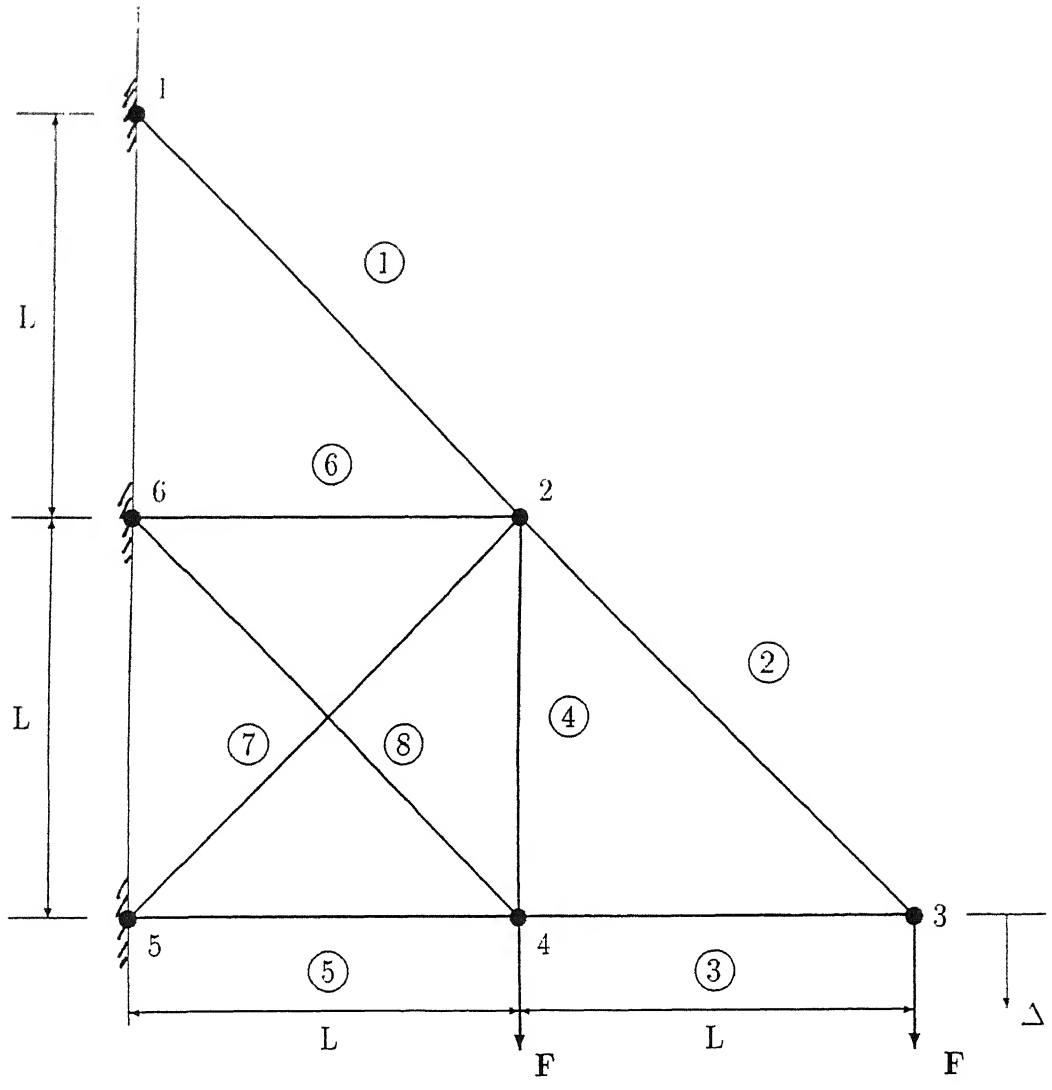


Figure 7.8: Eight-bar truss multiobjective optimization problem. The load value F = 100 kN, and L = 100 cm.

In this thesis, the topology is handled by removing an element, if its area of cross section is negligible. This way, by defining the negligible limit, the topological variables can be assigned to some schemata. These schemata represent strings belonging to the region between negligible area value and actual zero value. This may help in expediting the NSGA convergence towards Pareto-optimal regions. When an element's area falls below the limit, that element is removed and the feasibility of the structure is checked. If the truss is feasible, it is analyzed for stresses and outer node deflection. A bracket operator penalty function (Rao, 1991) is used to transform the constrained objective to a single function. The stress limits were taken as 14 kN/cm^2 in tension, and -8 kN/cm^2 in compression. The area bounds are taken as 0 cm^2 and 30 cm^2, and assumed that if an elemental area falls below $5cm^2$, that element should be removed. It was also observed that if any element's cross sectional area is less than $5cm^2$, the stresses induced in that element are greater than 2 to 3 times the limiting stresses. The Young's modulus for the material is assumed as $E = 2 \times 10^4$ kN/cm^2.

Using these values Koski (1988) repeated weighting method, with different weights, to find the Pareto-optimal points. This front is approximated with help of different Pareto-optimal values obtained with different weights. This front is used to compare NSGA's results. The parameters used in NSGA simulation are as follows

Maximum generation	:	250
Population size	:	200
String length (binary code)	:	64 (Eight 8-bit strings for each variable)
Probability of crossover	:	1.0
Probability of mutation	:	0.0
σ_{share}	:	30.0 (Calculated considering entire variable space, to induce 10 niches.)

Figures 7.9 through 7.12 show the population at different generations. Here the Pareto-optimal points shown in solid diamonds are taken from Koski (1988). The

Table 7.2: Typical topological optima found by NSGA. The areas are given in cm^2, volume in cm^3, and deflection in mm. Zero area indicates the absence of that member.

Design	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	Volume	Deflection
1	29.6	29.8	28.0	14.0	28.4	0	11.76	10.9	18676.8	1.52
2	28.7	24.2	29.4	7.7	29.64	9.7	9.05	0	16426.9	1.66
3	21.2	17.4	20.8	7.5	29.4	7.8	11.8	6.1	14584.1	2.08
4	19.6	21.7	15.6	10.5	23.7	7.6	0	7.17	12636.1	2.31
5	13.7	13.6	19.0	14.7	20.4	0	8.9	10.2	12012.1	2.32

representation of the performance space itself is very difficult as the topological objective is another implicit dimension. This can be observed from the table 7.2 that different topological designs have similar volume or deflection.

Figure 7.13 compares two NSGA populations at generation 100, obtained using different initial populations. These two populations converged to two different regions. This suggests to use NSGA with mutation. Figure 7.14 shows the population obtained by NSGA with mutation probability, $p_m = 0.01$, and cross over probability, $p_c = 0.8$. Here the distribution is more, but convergence is slower. The NSGA solutions are close to Koski results. One reason for deviation could be that only 8-bits (or 256 areas) are permissible in NSGA. Taking higher string length may provide closer results. Further possible extensions and modifications that make NSGA more robust are discussed in the next chapter.

7.5 Summary

This chapter presented successful application of NSGA to two real-world engineering optimization problems. Truss material minimization plays important role in minimizing its cost, but large structures will have undesirable displacements which are

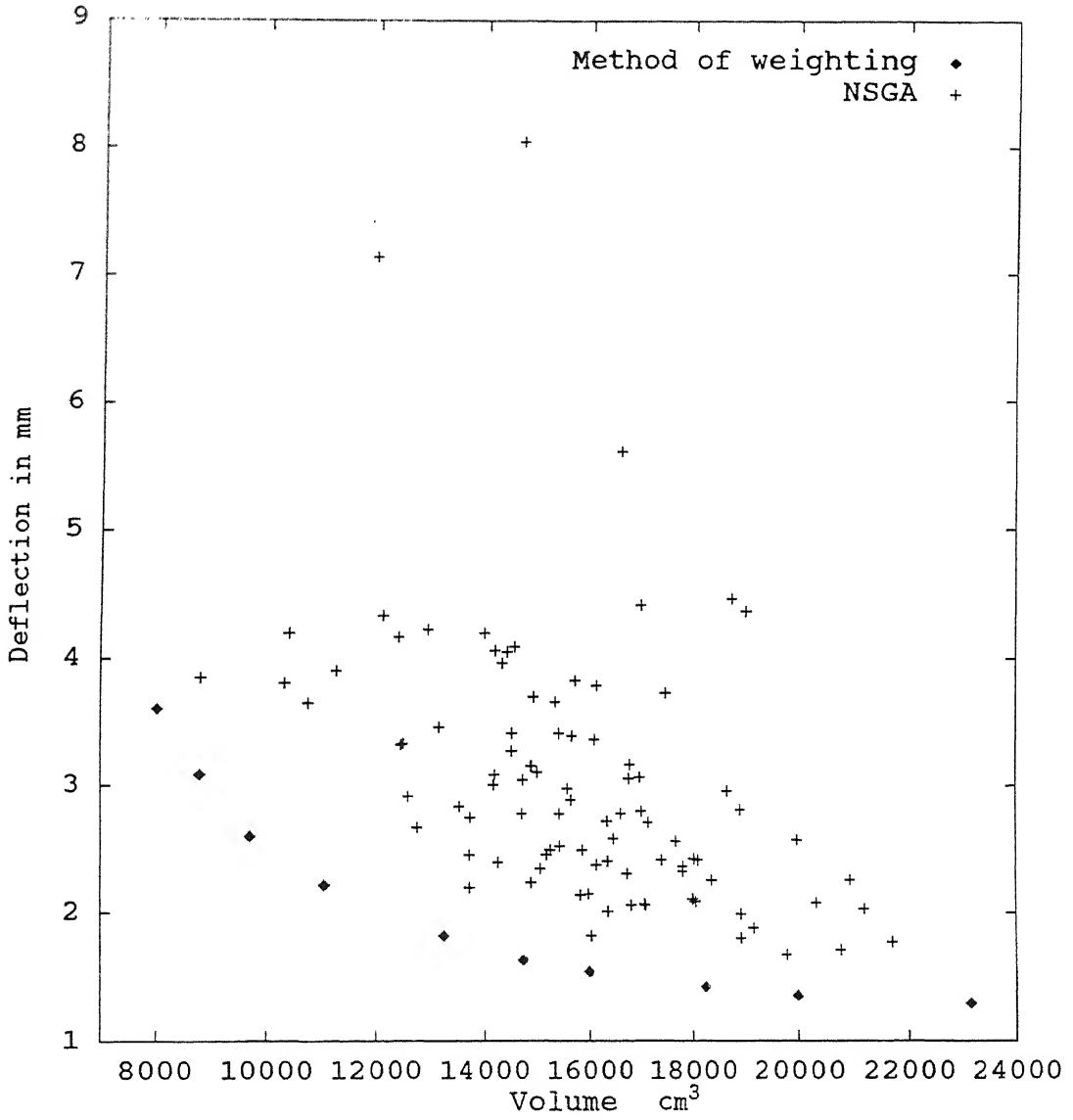


Figure 7.9: Population at generation 0 obtained using NSGA for eight-bar truss problem.

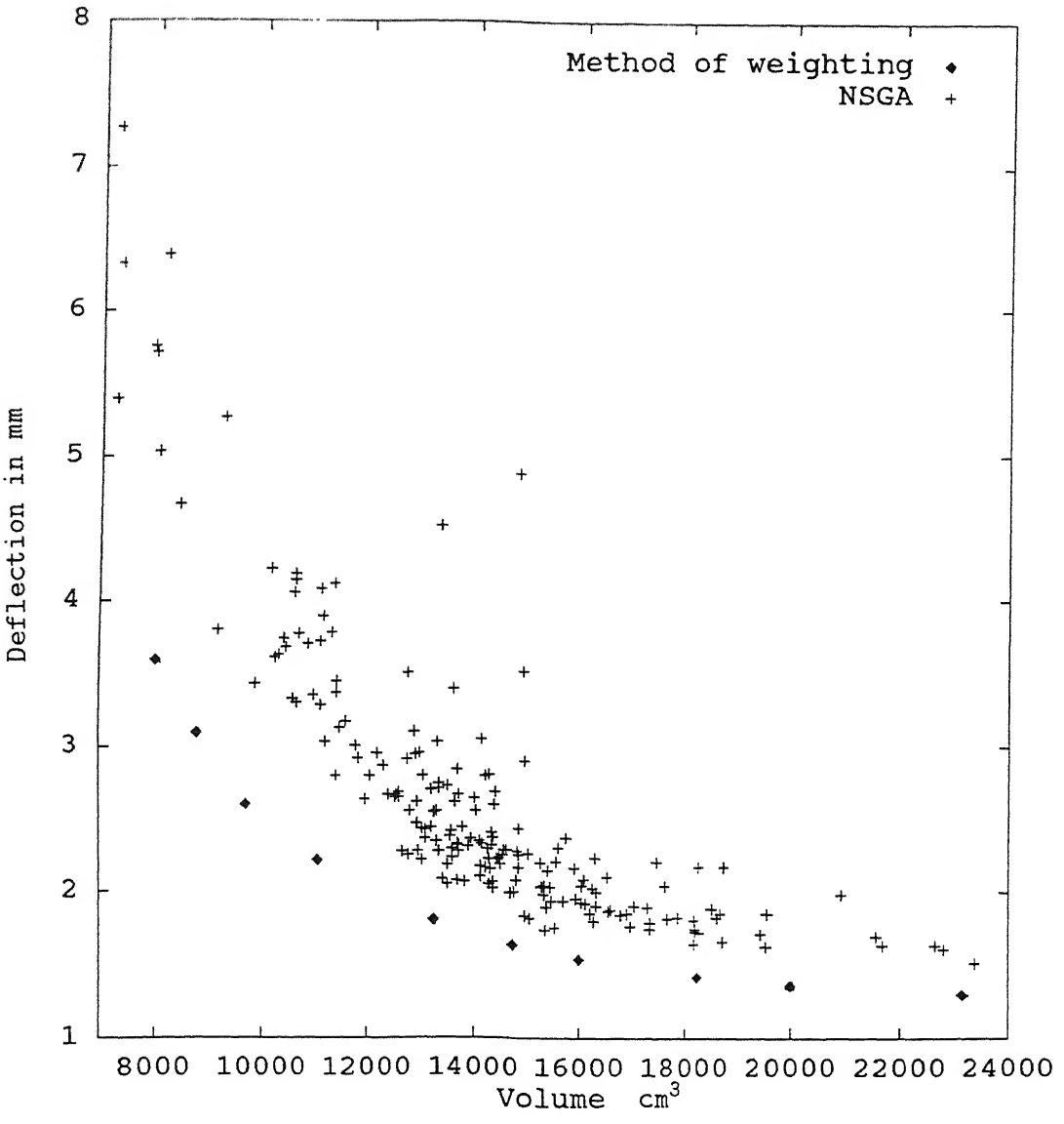


Figure 7.10: Population at generation 10 obtained using NSGA for eight-bar truss problem.

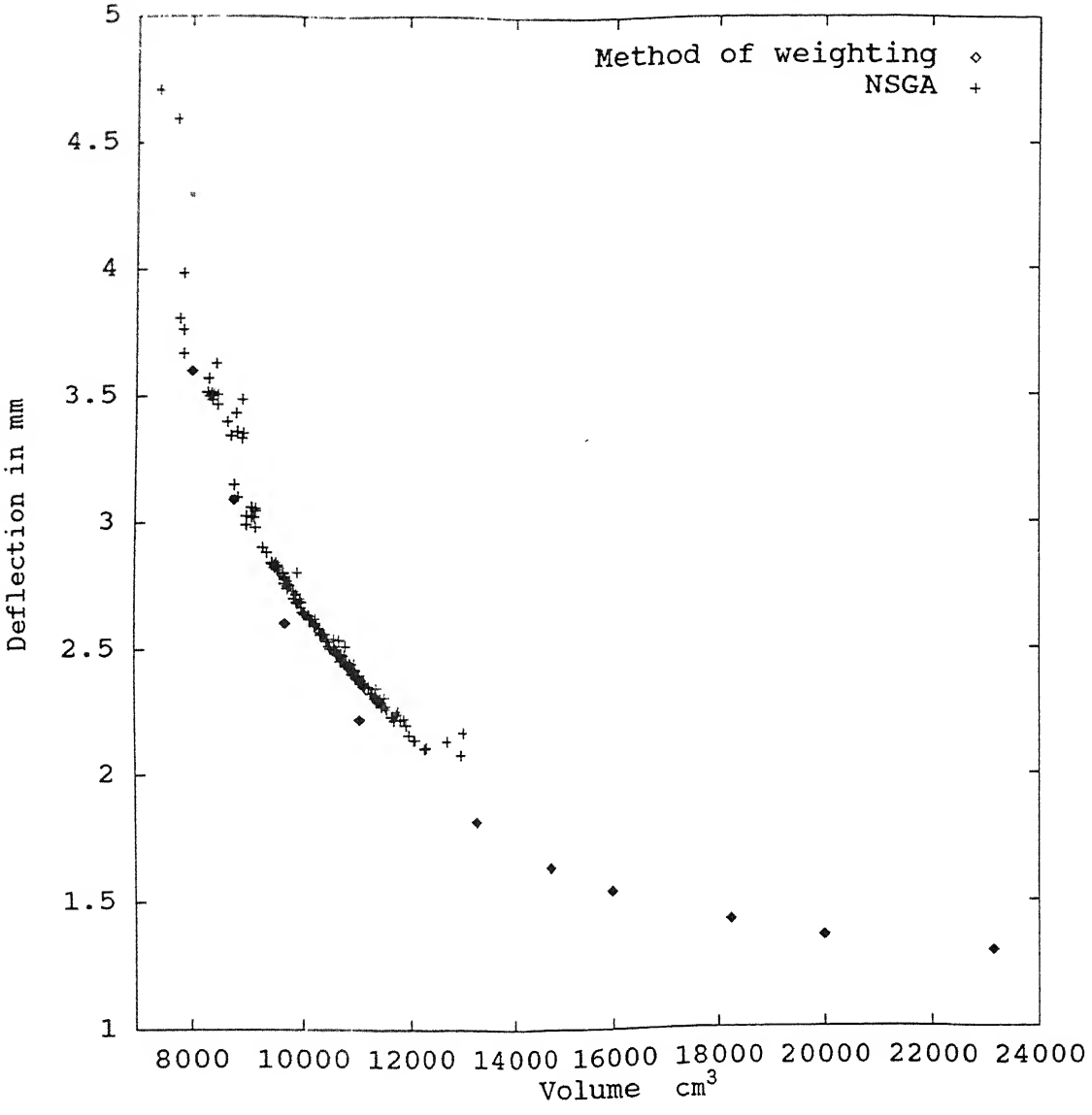


Figure 7.11: Population at generation 100 obtained using NSGA for eight-bar truss problem.

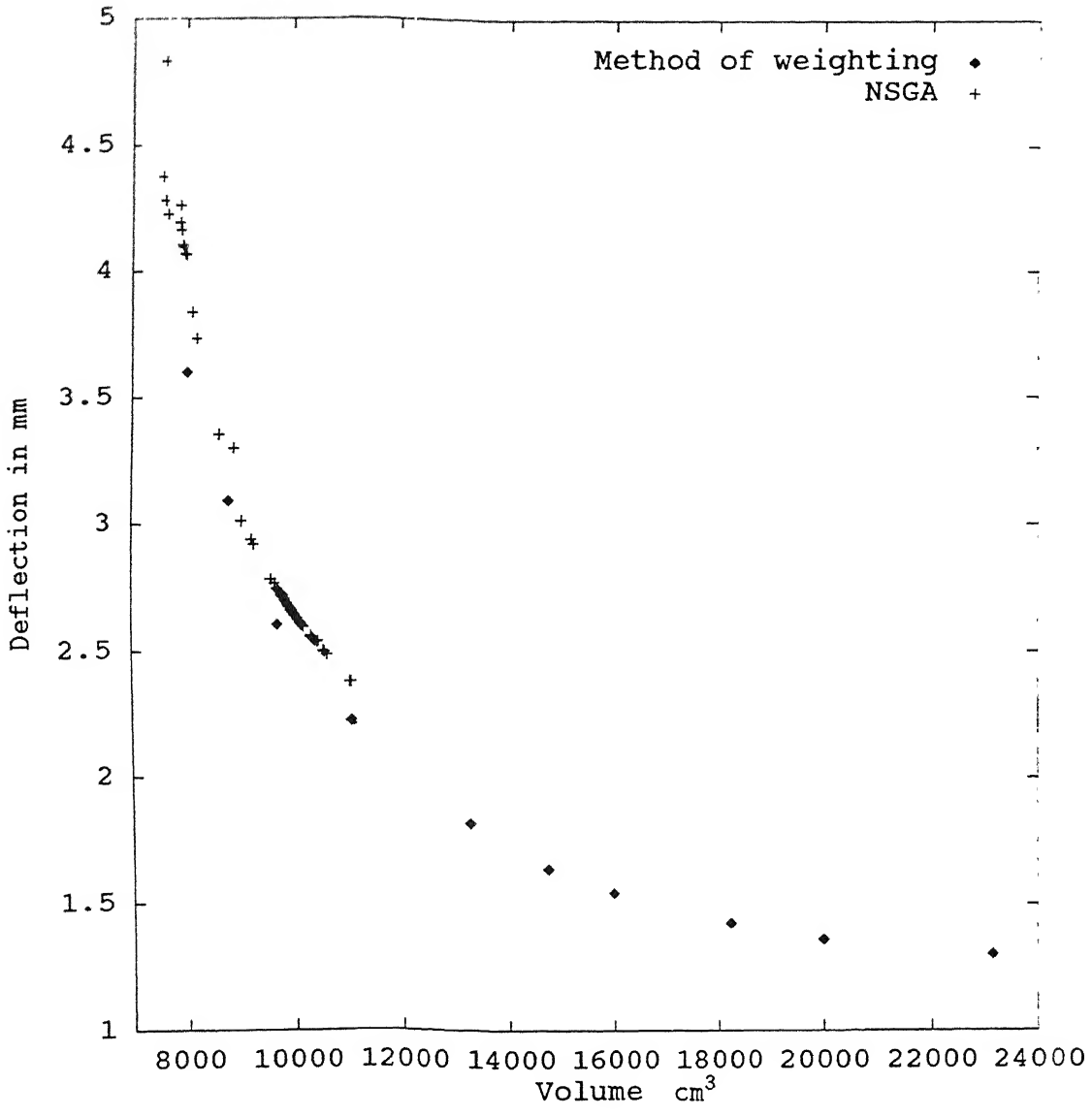


Figure 7.12: Population at generation 250 obtained using NSGA for eight-bar truss problem.

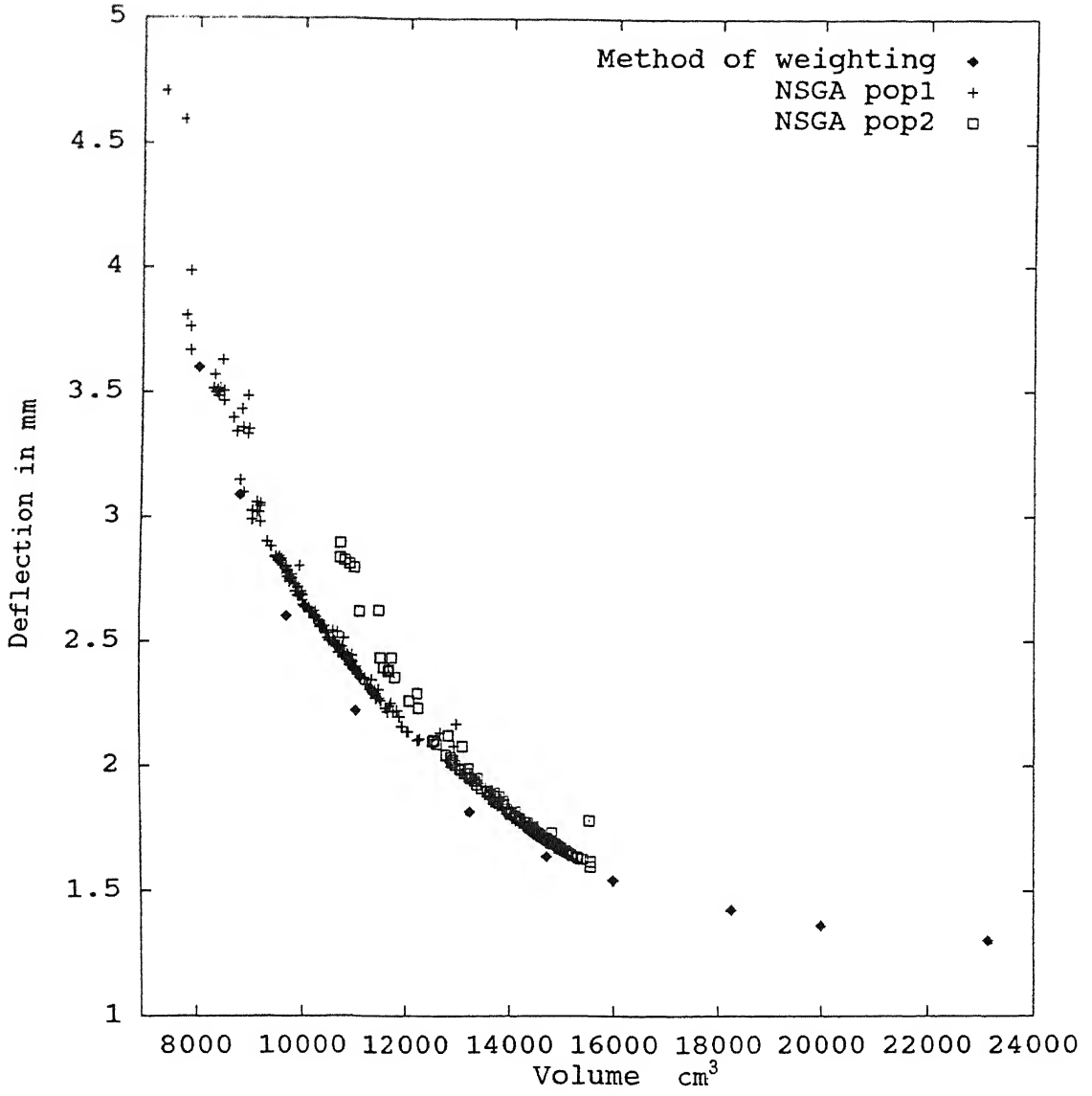


Figure 7.13: Comparison of population at generation 100 obtained using NSGA with different initial populations.

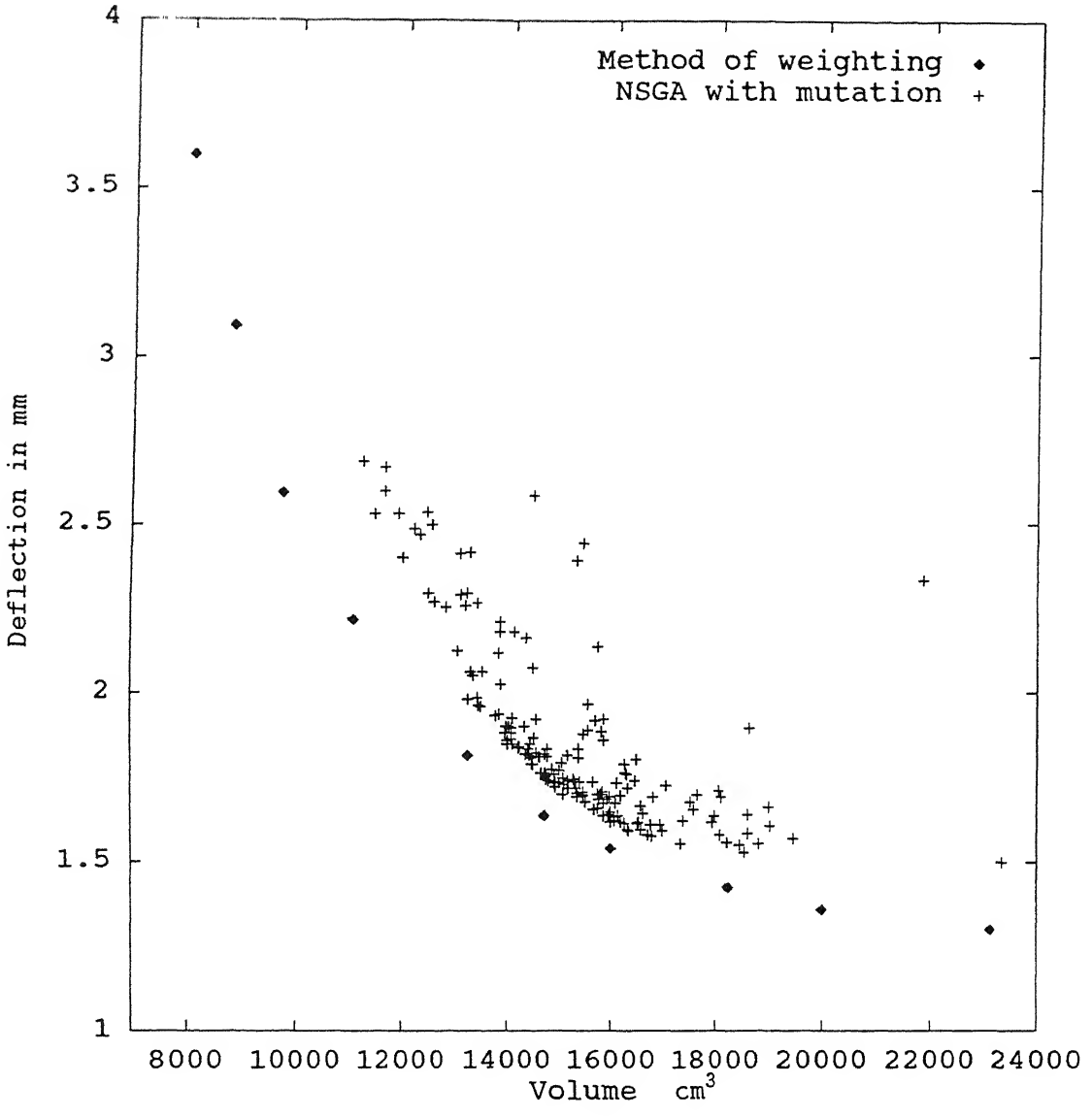


Figure 7.14: Population at generation 250 obtained using NSGA with mutation for eight-bar truss problem.

to be minimized. For an engineer it seems more advantageous to treat volume of a truss and critical displacements as independent objectives. He always desires a set of compromise solutions instead of single alternative. To achieve such a goal, NSGA may be used efficiently. In this chapter, two bicriteria truss optimization problems are considered. In the first problem, volume of a truss and a nodal deflection are considered as objectives, subjected to limiting stresses. The design variables are cross sectional areas of elements. NSGA is successful in distributing its population over a large portion in the nondominated fronts. The second problem, with similar objectives, but having additional variables in the form of topology changes, is solved by NSGA. Although this problem is more complicated, NSGA is able to find multiple Pareto-optimal points.

The results obtained show promising future of NSGA. Some possible extensions to this algorithm and future scope of research are discussed in the next chapter.

Chapter 8

Extensions

This thesis implemented the idea of nondominated sorting to solve multiobjective optimization problems. The results obtained show a promising future to this algorithm. Since this thesis implements NSGA for the first time, the robustness of NSGA can be questioned. Even this version of NSGA, with fitness sharing, proved to be much more robust than other GA approach VEGA. It can be modified and extended to solve wide variety of problems. This chapter discusses such possible extensions.

A number of extensions of this study can be pursued:

1. The σ_{share} parameter plays major role in distributing NSGA's population. This is because the number of niches induced in the variable space are fixed by this parameter. This thesis examined the effect of varying σ_{share} values. But detailed study in this area is an interesting avenue for further research.
2. Even though two objectives are used in both real-world problems, more objectives can be handled with NSGA. Moreover, the objectives need not be all minimization type, some of them could be minimization type and some could be of maximization type. In both situations, the definition of nondominated points will change, but the NSGA algorithm will remain the same. Unlike VEGA, the population size may not increase linearly with the number of objectives in NSGA. In VEGA, the population needs to

be changed linearly because the whole population needs to be divided to be used in selection for each objectives. Unless VEGA's population is increased, it is not possible to minimize the bias in its selection. With NSGA, the population size requirement may be more with number of objectives, but how the size requirement would increase is a matter of interesting future research.

3. The ranking selection in NSGA is implemented by introducing dummy fitness values. However, this will not control the selection pressure. To incorporate such control, proper fitness scaling method (Goldberg, 1989) needs to be used. By scaling the dummy fitness values, the best nondominated point can be set to get required number of copies, while setting the worst dominated point's copies to zero. This way selection pressure can be controlled. Another selection approach that can be used is discussed below.

4. It has been found elsewhere (Goldberg and Deb, 1991) that *tournament* selection puts a more controlled selection pressure and has a faster convergence than proportionate selection method used in this study. The niching technique suggested by Oei, Goldberg and Chang (1992) can be tried with tournament selection to replace sharing and proportionate selection in NSGA for more controlled and hopefully faster solutions.

Finally, other advanced operators like inversion, multipoint crossover etc., can be incorporated. Schemes such as mating restriction (Goldberg, 1989), used in conjunction with niche and speciation techniques, may enhance the robustness of NSGA.

Chapter 9

Conclusions

Multiobjective optimization is different from single-objective optimization, in the sense that no unique solution exists in the former case, rather the solution comprises a set of possible alternatives, known as nondominated or Pareto-optimal points. Even though there exists a number of classical multiobjective optimization techniques. they require some a priori problem information. Most of these algorithms scalarize an objecive vector using weights or goals, resulting in a single point solution to the problem. Often times the obtained solution may not be a Pareto-optimum solution. This thesis have overviewed three such methods that are commonly used in traditional multiobjective optimization.

Since genetic algorithms use a population points, they are suitable to find multiple Pareto-optimal solutions simultaneously. Schaffer's vector evaluated genetic algorithm (VEGA) was an early effort along this direction. But his VEGA suffered due to its bias against some middling nondominated points. Its individual selection resulted in a population drift towards individual optima of objectives, instead of finding maximum possible Pareto-optima points. Experimental results presented in this thesis, similar to what Schaffer had found, have shown this drawback.

A new approach, a nondominated sorting genetic algorithm suggested by David Goldberg, has been described and implemented in this thesis. Three test problems

used in other studies have been selected. Simulation results on three test problems have shown that this algorithm (named as NSGA) can maintain stable and uniform reproductive potential across nondominated individuals, which is a serious drawback of VEGA. The power of NSGA lies in the successful transformation of a multiobjective problem, no matter how many objectives are there, to a single function problem without loosing the concept of vector optimization. This enables NSGA to identify the building blocks that represent Pareto-optimal regions. The results suggest that NSGA can be used to find different Pareto-optimal solutions, the knowledge of which could be very useful to the designers or decision makers. The successful application of NSGA to two engineering multiobjective optimization problems suggests immediate application of NSGA in more complex engineering problems. A number of suggestions for immediate extension and application of NSGA to several multiobjective optimization problems has also been discussed.

References

Bathe, K. J. (1990). *Finite element procedures in engineering analysis.* New Delhi: Prentice Hall of India.

Chandrupatla, T. R. and Belegundu, D. A. (1991). *Introduction to finite elements in engineering.* New Delhi: Prentice Hall of India.

Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective decision making theory and methodology.* New York: North-Holland.

Deb, K. (1989). *Genetic algorithms in multimodal function optimization.* Masters thesis, University of Alabama. (TCGA Report No. 89002). The Clearinghouse for Genetic algorithms, University of Alabama. Tuscaloosa.

Deb, K. and Goldberg, D. E. (1991). An Investigation of niches and species formation in genetic function optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer(Ed.)(pp 42-50). San Mateo, CA: Morgan Kaufman.

Deb, K. (1993). Genetic algorithms for engineering design optimization. *Advanced Study Institute on Computational Methods for Engineering Analysis and Design*. J. N. Reddy, C. S. Krishnamoorthy, and K. N. Seetharamu (coordinators) (pp 12.1-12.25). Indian Institute of Technology, Madras.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, NY: Addison-Wesley.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*(69-93). San Mateo, CA: Morgan Kaufman.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette (Ed.) (41-49).

Hans, A. E. (1988). Multicriteria optimization for highly accurate systems. *Multicriteria Optimization in Engineering and Sciences*, W. Stadler(Ed.), Mathematical concepts and methods in science and engineering, 19, 309-352. New York: Plenum press.

Holland, J. H. (1975) *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Krishnamoorthy, C. S. and Rajeev, S. (1993). Structural optimization using genetic algorithms. *Advanced Study Institute on Computational Methods for Engineering Analysis and Design*. J. N. Reddy, C. S. Krishnamoorthy, and K. N. Seetharamu (coordinators) (pp 13.1-13.21). Indian Institute of Technology, Madras.

Oei, C. K. Goldberg, D. E. and Chang, S. (1991). *Tournament selection, niching and the preservation of diversity*, (IlliGAL Report No: 910011). University of Illinois, Urbana.

Rao, S. S. (1991). *Optimization theory and application*. New Delhi: Wiley Eastern Limited.

Rao, S. S. Sundararaju, K. Prakash, B. G. and Balakrishna, C. (1992). Fuzzy goal programming approach for structural optimization. *Journal of Structural Optimization*, 30, 5, 1425-1432.

Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties*. Doctoral dissertation, University of Michigan. *Dissertation Abstracts International, 28*(7) 2734 B. (University Microfilms No:67-17,836).

Schaffer, J. D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms*. Doctoral dissertation, Vanderbilt University, Electrical Engineering, Tennessee. (TCGA file No. 00314).

Tamura, K. and Miura, S. (1979). Necessary and sufficient conditions for local and global nondominated solutions in decision problems with multiobjectives. *Journal of Optimization Theory and Applications*, 27, 509-523.

Vincent, T. L. and Grantham, G. J. (1981). *Optimality in parametric systems*. New York: John Wiley and sons.

ME-1994-M-SRI-MUL.